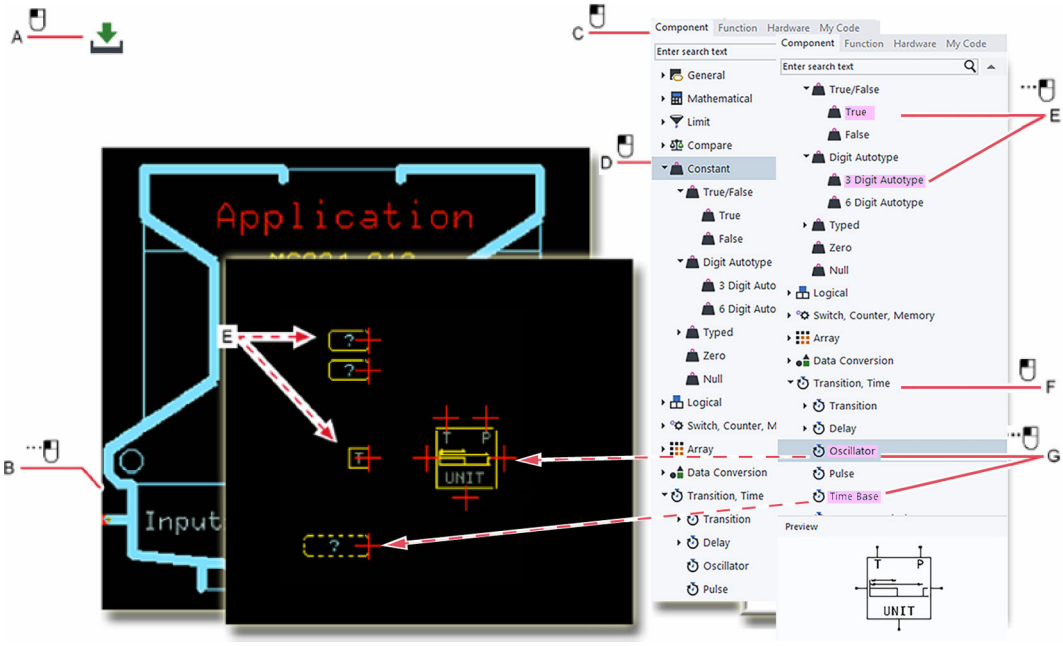
ENGINEERING
TOMORROW

User Manual

PLUS+1® GUIDE
Software



www.danfoss.com

Revision history

Table of revisions

Date	Changed	Rev
January 2024	Supports 2024.1	3001
September 2023	Supports 2023.3	2901
June 2023	Supports 2023.2	2801
March 2023	Supports 2023.1	2701
October 2022	Supports 2022.3	2601
June 2022	Supports 2022.2	2501
March 2022	Supports 2022.1	2401
June 2021	Graphics corrections	2302
February 2021	Updated to support 12.2	2301
June 2020	supports 12.1 and later	2201
June 2020	Changed document number from 'AQ00000026' and '10100824' to 'AQ152886483724'	2102
October 2019	supports 12.0 and later	2001
September 2019	Major update - supports 12.0 and later	1901
May 2019	Major update - supports 11.1 and later	1901
October 2018	Major update - supports 11.0 and later	1801
May 2018	Major update - Supports 10.1x and later	1703
November 2017	Major update - Supports 10.0x and later	1601
April 2017	Major update - Supports 9.1.x and later	1503
December 2016	Minor revision to document layout only	1502
October 2016	Major update - Supports 9.0.x and later	1501
February 2016	Major update - Supports 8.0.x and later	1401
August 2015	Major update - Supports 7.2.x and later	1300
December 2014	Supports 7.1.x and later	MC
December 2013		LA

Contents

Contents

Contents

Contents

Screen Editors

Contents

Application Data Logging

Contents

Support Tools

Introduction

Introduction

About this manual

This manual describes the user interface and programming related information of the PLUS+1® software.

Some functions require an add-on license. The functions are described in respective section.

To buy an add-on license, see PLUS+1® Software License Manager Help, **AQ152886482086**.

For short information about the add-on functions, see Data Sheet PLUS+1® GUIDE Add-on licenses, **AI00000253**.

Intended use

In PLUS+1® GUIDE you build your application with drag-and-drop logical components or software blocks in the graphical interface. It is engineered to accelerate your development process and bring higher quality machines to market faster.

Use the PLUS+1® Service Tool to download the built application to a controller. You can download parameters to a controller, then you can log and tune the controller performance.

Safety definitions and reading information

Safety definitions

Warning information

Warning information points out potential dangers which can, if the warnings are not followed, result in personal injury or product damage.

Warning

Situation that may result in death or serious personal injury if the instruction is not followed.
Do not to proceed until all stated conditions are met and clearly understood.

Caution

Situation that may result in damage to the product if the instruction is not followed.
Do not to proceed until all stated conditions are met and clearly understood.

Important information

Important information that is marked as below shall facilitate the work process, operation/handling or increases understanding of the information.

Information that is important without being safety related.

Reading information

Software items are identified by bold text: **Manager panel**.

To separate menu levels an arrow is used: **File** > **Open** refers to the **File** menu and the **Open** command.

Code and file paths are identified by this font: `\Danfoss\PLUS1\`.

Introduction

Command name are identified by this font: `Ctrl+D`.

Risk reduction

It is important to design, test and secure applications developed with the PLUS+1® GUIDE software to reduce the risk of personal injury and equipment damage.

The applications that you create with the PLUS+1® GUIDE software typically control heavy, powerful, and mobile off-road equipment such as tractors, cranes, and harvesters. Under normal operating conditions, using this type of machinery always involves the risk of personal injury and equipment damage. Abnormal operating conditions greatly increase the risk of personal injury and equipment damage.

The PLUS+1® GUIDE software has no automatic protections against these risks. The tool has no protection against the risks that result from bugs in the tool software, errors in the tool manual, or incompatibilities between software versions of the tool.

Warning

You must design and test your application to reduce these risks. Secure your application against unauthorized changes in its operating parameters to reduce these risks.

Design

You have the responsibility when you design a PLUS+1® GUIDE application to include the fault checking and the error handling needed to reduce risks in normal and abnormal operating conditions. The following are some items to consider when developing fault checking and error handling for your application:

- How the machine is normally used.
- Possible operator errors and their consequences.
- Industry safety standards and legal requirements.
- Input and output failures and their consequences. These failures can include:
  - Joystick, sensor, and other inputs suddenly going to ±100 % or to 0 %.
  - Outputs that control machinery direction, speed, and force suddenly changing direction or going to ±100 % or to 0 %. Decide how likely each failure is. The more likely a failure, the more you need to protect against the consequences of the failure.
- The sequence of events and consequences of a fault or error.
- The sequence of events and consequences of an emergency stop.

Test

You have the responsibility once you have created an application to test the application. You should download your application to hardware and test its operation under both normal and abnormal operating conditions. You should make sure that:

- Individual inputs produce expected outputs.
- Fault handling and error checking work as designed.

You must repeat your tests whenever you make configuration, calibration, or software changes to your application.

Secure

- You have the responsibility to secure your application against unauthorized changes.
- You should always use the PLUS+1® GUIDE program's Tool Key feature (or a parameter PIN feature for PLUS+1® C Open) to restrict access to your application's operating parameters.

Risk reduction

- Without Tool Key or PIN protection, there is an increased risk that unauthorized personnel could use the PLUS+1® Service Tool program to change your application's operating parameters
- Tool Key/PIN protection reduces the risk that unauthorized personnel could use the PLUS+1® Service Tool program to change your application's operating parameters.

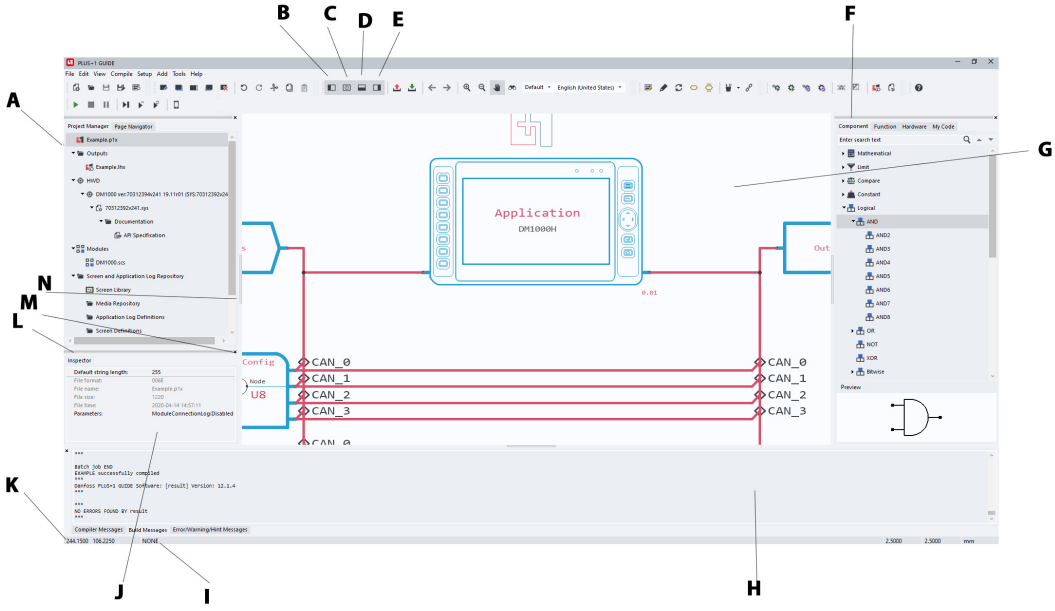
Warning

Changes in your application's operating parameters could cause unexpected machinery movement that result in personnel injury and equipment damage.

User Interface

Major elements of PLUS+1® GUIDE user interface, including menus, toolbar, dialogs, and tabs.

Overview



A	**Manager** Panel: Manage protect files, navigate project pages
B	Displays the **Manager** panel
C	Displays the **Inspector** panel
D	Displays the **Compiler** panel
E	Displays the **Selector** panel
F	**Selector** panel: Select hardware, components, functions
G	Design Area: Create your application here
H	**Compiler** panel: View compiler messages
I	Command selection
J	**Inspector** panel: View and change values
K	Crosshair position
L	Drag to unlock panels
M	Click to close panels
N	Click to lock panels

The **Manager** tab (on the left of this window) has **Project Manager** and **Page Navigator** tabs:

- Use the **Project Manager** tab to display the files installed from the **Hardware** tab.
- Use the **Page Navigator** tab to navigate you through the pages in the application.

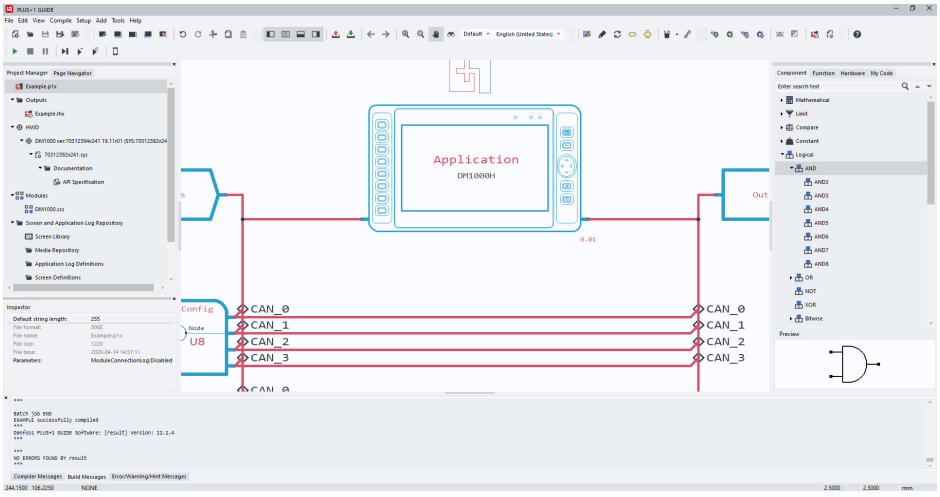
The Selector tab (on the right of this window) has **Component**, **Function**, **Hardware** and **My Code** tabs:

User Interface

- Use the **Component** tab to select the basic components used within in your application, such as **AND** gates.
- Use the **Function** tab to select the advanced functions used within your application, such as a **Soft Ramp**.
- Use the **Hardware** tab to select the files needed to create and compile an application for different models of PLUS+1® hardware.
- Use the **My Code** tab to store code that you want to reuse in multiple projects.

The **Compiler** tab (at the bottom of this window) displays compiler messages.

You can close, resize, and undock these tabs.



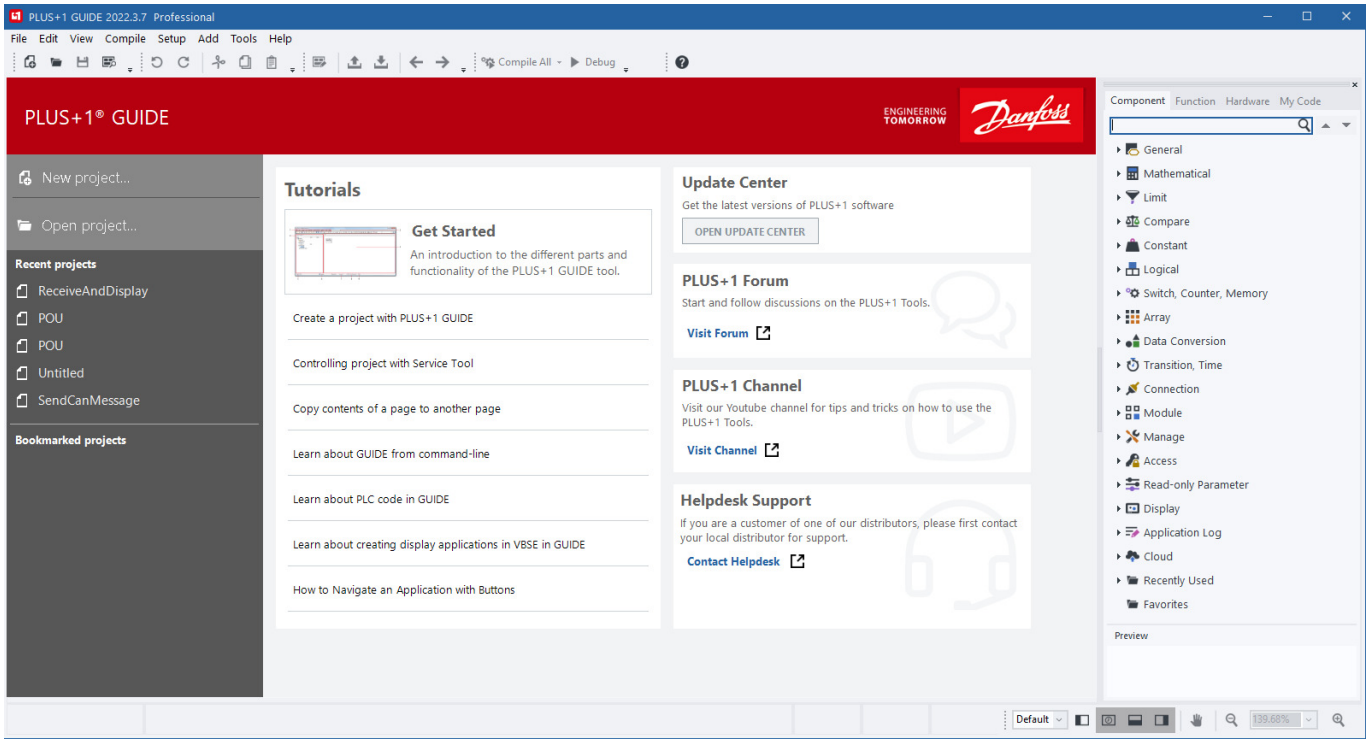
PLUS+1® GUIDE window

Item	Description
Menus	Use the menu bar to access PLUS+1® GUIDE commands and information.
Toolbar	Use the buttons in the toolbar to access common PLUS+1® GUIDE commands and information.
Selector tab	• **Component** tab — shows a tree view of components that you can drag into the Drawing Area when creating your application. • **Preview** tab — displays a preview of the selected component. • **Function** tab — shows a tree view of functions that you can drag into the **Drawing Area** when building your application module. • **Hardware** tab — shows a tree view of Hardware Descriptions for each PLUS+1® hardware model. These Hardware Descriptions link to the resources needed by the PLUS+1® GUIDE software to create and compile applications for specific PLUS+1® hardware models. • **My Code** tab — shows a tree of your own stored reusable code, organized by the type of code. Use the **Quick Search Field** to easily find what you are looking for in any of these 4 tabs. The **Quick Search Field** implements case-insensitive search with wrap-around behavior.
Manager tab	• **Project Manager** tab — shows a tree view of the current project data, including the selected Hardware. Note: You must install at least one Hardware Description file before you can create an application. • **Page Navigator** tab — shows a tree view of all the pages within an application module. Click to display a selected page in the **Drawing Area**.
Drawing Area	Create your application module here.
Inspector tab	Use this tab to view and change the properties of items that you select.
Compiler Messages tab	• **Build Messages** tab — displays messages from the compiler as it compiles your application into a downloadable LHX format file. • **Error/Warning/Hint Messages** tab — displays details about errors in the compile process. • **Compiler Messages** tab — displays messages sent to the compiler log file.

User Interface

Start Page

The start page is available by default when no project is open in GUIDE. It can be disabled in **Options** under *Project Open/Close Settings* on page 68.



Item	Description
New Project...	Displays the **Create New Project** dialog. Use this dialog to create a new project and to select a folder for the project files.
Open Project...	Displays the **Open Project** dialog. Use this dialog to locate and open P1P and P1X project files.
Recent projects	A list of recently opened projects. Select a project to open from this list.
Bookmarked projects	A list of bookmarked projects. Select a project to open from this list.
Tutorials	A list of tutorial links into the HTML version of this manual.
Update Center	Shortcut to start the PLUS+1® **Update Center**.
PLUS+1® Forum	Shortcut to the PLUS+1® Forum online.
Helpdesk Support	Contact information to PLUS+1® Helpdesk.

Languages

GUIDE supports two languages; English and Chinese.

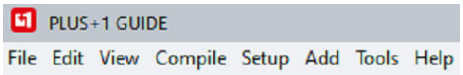
English is the default language and is available everywhere in the tool.

Chinese is supported in menus, toolbars, manuals, and most dialogs. Most of the Component tree and My Code tree are also translated.

Switch between languages by using the drop-down list. It is also possible to switch to the default language (English) by keyboard shortcut (Ctrl+Alt+F by default).

User Interface

Menus



For more information see the topics:

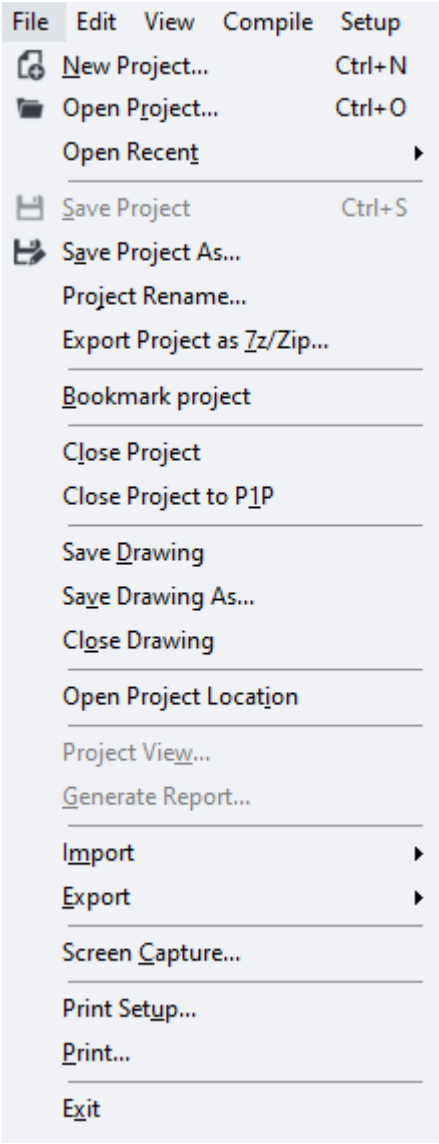
File Menu on page 17.
Edit Menu on page 20.
View Menu on page 22.
Compile Menu on page 24.
Setup Menu on page 25.
Add Menu on page 26.
Tools Menu on page 27.
Help Menu on page 28.

Toolbar buttons duplicate commonly used menu commands. For more information see the chapter *Toolbar* on page 29.

User Interface

File Menu

Manage project files with the **File** menu.



File menu description

Item	Description
New Project...	Displays a **Create New Project** window. Use this window to name a new project and to create a folder for the project files.
Open Project...	Displays an **Open Project** window. Use this window to locate and open P1P and P1X project files.
Open Recent	Displays a list of recently saved projects. Select a project to open from this list.
Bookmarked Projects	A list of bookmarked projects. Select a project to open from this list.
Save Project	Saves the current project files without zipping them into a single P1P format file.
Save Project As...	Displays a **Save Project As** window. Use this window to save the current project under a new name and in a new folder.
Project Rename...	Displays a **Project Rename** window. Use this window to change the name of the current project.
Export Project As 7z/ Zip...	Displays a **Save As** window. Use this window to save your project files in the 7z or ZIP compressed archive file format.

User Interface

File menu description (continued)

Item	Description
Bookmark Project	Add the current project to the list of Bookmarked projects. If the current project is already Bookmarked, this menu item changes to "Remove from Bookmarks." This functionality is also available from the context menu of the P1X node in the **Project Manager**.
Close Project	Closes the current project without exiting the PLUS+1® GUIDE software. Displays a **Project Modified** window (if you have modified your project) and a **Compressed Format** window. • Use the **Project Modified** window to choose to save any changes that you have made in your project. • Use the **Compressed Format** window to choose to compress project files into single P1P format file.
Close Project to P1P	Closes and compresses the current project into the P1P file format without exiting the PLUS+1® GUIDE software. Displays a **Project Modified** window (if you have modified your project) and then closes the project and compresses its files into the P1P file format. Use this window to choose to save any changes that you have made in your project.
Save Drawing	Saves the application drawing as an SCS file, using the module name.
Save Drawing As...	Displays a **Select/Define New Job File Name** window. Use this window to save the application drawing as an SCS file, under a new name.
Close Drawing	Closes the current application drawing without closing the project.
Open Project Location	Opens the project folder in Windows Explorer. The project P1X file will be selected in Windows Explorer.
Project View...	Displays a **Project View** window. Use this window in combination with a third-party version control program such as Apache Subversion® to: • Identify the core files that you need to open, continue, and compile a project. • Export these core files for safekeeping to your version-control repository.
Generate Report...	Displays a **Save As** window. Use this window to save a TXT format file that reports project-related items such as the: • Project path • Names of project files — These are the core files that you need to open, continue, and compile a project. — If you are using version control, you must export these files for safe keeping to your version control repository. • Names of temporary project files • Names of non-project files • Names of referenced files
Import	• **Import Block** — displays a **PLUS1 Editor Load Block** window. — Use this window to select and import an SCS file into the Drawing Area. When you import two or more identical blocks, the link feature automatically links the contents of the blocks. • **Import Page** — displays a **Select Job File Name** window after you select a page in the PLUS+1® GUIDE window's Drawing Area. — Use this window to select and import an SCS file into the Drawing Area. When you import two or more identical blocks, the link feature automatically links the contents of the blocks. The **Import Block** and **Import Page** commands do not work with **Define Areas** and **Define Screen** blocks.
Export	• **Export Block** — displays a **Symbol Block Export Binary** window after you select items for export in the Drawing Area. — Use this window to export the selected items to an SCS file only from the module in which you are working. — External modules, referenced through **Module** components (such as the **Call Module** component) do not export. • **Export Page** — displays a **Select/Define New Job File Name** window after you select a page in the PLUS+1® GUIDE window's Drawing Area. — Use this window to select an SCS file and export the selected page to an SCS file in the Drawing Area. Command exports items only from the module in which you are working. — External modules, referenced through **Module** components (such as the **Call Module** component) do not export. The **Export Block** and **Export Page** commands do not work with **Define Areas** and **Define Screen** blocks.
Screen Capture	Displays a **Select Screen Dump Format** window after you select items in the PLUS+1® GUIDE window's Drawing Area. Use this window to choose to print the selected items, save them to a BMP or TIFF file, or copy them to the clipboard in a BMP or Metafile format.
Print Setup	Displays a **Print Setup** window. Use this window to select a printer and set its properties.

User Interface

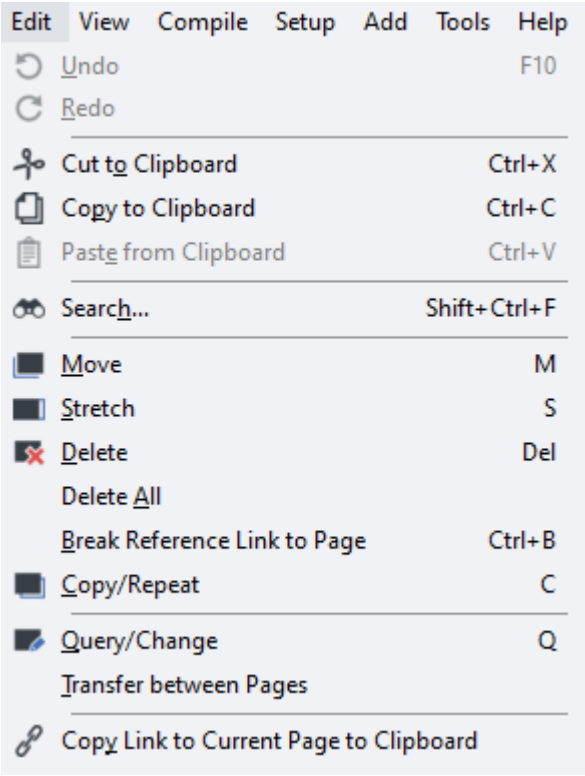
File menu description (continued)

Item	Description
Print	Displays a **Print** window. Use this window to print pages from the application.
Exit	Closes the PLUS+1® GUIDE software. Closing the application displays a **Project Modified** and then a **Save in P1P** windows. • Use the **Project Modified** window to choose to save or discard changes that you have made in your project. • Use the **Save in P1P** window to compress project files into a single P1P file.

User Interface

Edit Menu

The **Edit** menu commands change and modify items in the drawing area.



Edit menu description

Item	Description
Undo	Reverses programming actions. Depending on computer memory, you can undo up to 10 actions.
Redo	Reverses **Undo** commands. Depending on computer memory, you can reverse up to 10 Undo commands. The **Options** window enables and disables the undo/redo function. To display this window, click **Options** in the **Setup** menu. For more information, see *Setup Menu* on page 25
Cut to Clipboard	Deletes items selected in the Drawing Area and copies them to the Clipboard.
Copy to Clipboard	Copies items selected in the Drawing Area to the Clipboard.
Paste from Clipboard	Pastes the contents of the Clipboard into the Drawing Area. The link feature automatically links the contents of duplicate pages that are copied from the Clipboard.
Search	Search/Replace items in your current project.
Move	Moves items selected in the Drawing Area.
Stretch	Stretches segments in selected routes and moves selected items. Click a route to add a vertex to the route. The new vertex can then be selected and moved without changing the positions of other vertexes in the route.
Delete	Deletes items selected in the Drawing Area. Selected items turn white. • An **Attributes** window displays when you select a single item or identical items. Click **OK** to delete your selection. • A **Select Item Class** window displays when you select different items, that you want to delete: — asterisks (*) identify items that will be deleted — dashes (–) identify items that will not be deleted • Click **OK** to delete your selections.
Delete All	Displays a **Question** window with a **Delete All Items?** message. Click **Yes** to delete all items in the Drawing Area.
Break Reference Link to Page	Displays a **Break Link** window when you select a linked page in the in the Drawing Area. Click **Yes** in this window to break the selected page's links to other pages.

User Interface

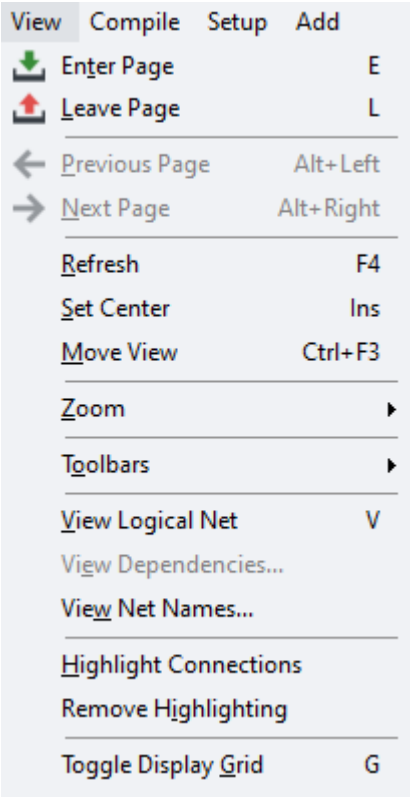
Edit *menu description (continued)*

Item	Description
Copy/Repeat	Copies items selected in the Drawing Area. Selected items turn white. • Click to place copied items. • Press **Esc** to stop placing copied items. This command only works within the Drawing Area. The link feature automatically links copied pages.
Query/Change	Use to change the properties (such as data type, text, and function) of items in the Drawing Area. Click an item whose property you want to change. A dialog box appropriate for the selected item displays. Change the property in this window.
Transfer between Pages	Cut the selected contents of a source page into a destination page. The destination can be any page in the application. After moving the source page into the destination page, you can choose either to continue working in the destination or to return to the source page.
Copy Link to Current Page to Clipboard	Copy the PLUS+1® link to the current page to clipboard.

User Interface

View Menu

The **View** menu commands change the view in the Drawing Area.



View menu description

Item	Description
Enter Page	Enters a selected page. To enter a page, click within the page boundaries or drag at a page port.
Leave Page	Leaves the current page.
Previous Page	Moves one page backward through your page navigation history.
Next Page	Moves one page forward through your page navigation history.
Refresh	Refreshes the Drawing Area.
Set Center	Centers the Drawing Area on where you click the pointer.
Move View	Displays a movable, transparent rectangle with white borders. Click to center the Drawing Area within the borders of this rectangle.
Zoom	• **In** – zoom in and center the Drawing Area on where you click the pointer. Alternative: Right-click and drag diagonally to the upper right. • **Out** – zoom out and center the Drawing Area on where you click the pointer. Alternative: Right-click and drag diagonally to the lower left. • **Area** – zoom the Drawing Area on an area defined by two pointer clicks. • **Fit Page** – sizes the view to fit all items on the page into the Drawing Area. • **100%** – zooms the Drawing Area to midway between min. and max. views. Press Home to fit the entire page in the Drawing Area. A thin blue line (—) indicates the boundaries of a page.
Toolbars	Toggles the display of button sets for the **File**, **Edit**, **View**, **Compile**, **Add** and **Help** functions.
View Logical Net	Displays a **Logical Net** window. Use this window to trace the connections of a selected logical net throughout a single page or multiple pages.

User Interface

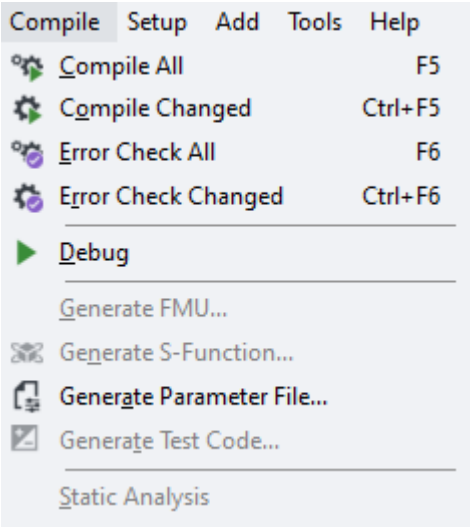
View menu description (continued)

Item	Description
View Dependencies...	Shows dependency between signals in the software and shows it directly in the code.
View Net Names...	Displays various windows that show floating entries, interface port mismatches, and net errors. Click items in these lists to see previews of the pages where these problems are located.
Highlight Connections	Highlights items selected in the Page Interface Editor window's Drawing Area.
Remove Highlighting	Remove highlighting from items in the **Page Interface Editor** window's Drawing Area.
Toggle Display Grid	Turn the display grid off and on. The display grid is the grid that you see in the Drawing Areas of the PLUS+1® GUIDE, **Page Interface Editor**, and **Module Viewer** windows.

User Interface

Compile Menu

The **Compile** menu commands control the compile functions.



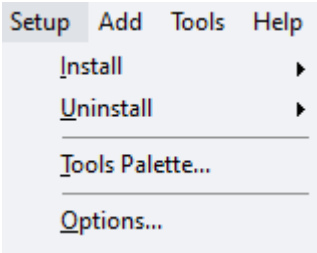
Compile menu description

Item	Description
Compile All	Compiles all application modules to produce a downloadable file.
Compile Changed	Compiles only the modules that you changed since the last **Compile All**. Use this command to save compile time when you have more than one module.
Error Check All	Compiles to check for errors in all modules. The **Error Check All** and **Error Check Changed** commands do not produce downloadable files.
Error Check Changed	Compiles to check for errors only in the modules that you changed since the last **Compile All**.
Debug	Prepare the application for debugging.
Generate FMU...	Use the Generate FMU functionality to export a PLUS+1 GUIDE page or module to a Functional Mock-up Unit (FMU) according to version 2.0.1 of the Functional Mock-up Interface standard. Refer to https://fmistandard.org/.
Generate S-Function...	Compiles the current page into a *Simulink S-Function*.
Generate Parameter File...	Use **Generate Parameter File** to generate Read-only parameter files for the application.
Generate Test Code...	Generates test code for pages with defined tests.
Static Analysis	Perform static analysis of the PLUS+1® application.

User Interface

Setup Menu

The **Setup** menu commands set up the PLUS+1® GUIDE application programming environment.



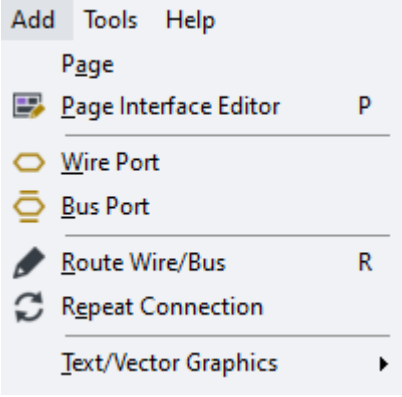
Setup menu description

Item	Description
Install	Displays an **Install Hardware** window. Use this window to browse to and add hardware definitions to the **Hardware** tab. Displays an **Install Function Library** window. Use this window to browse to and add library files to the **Library** tab. Displays an **Install Font** window. Use this window to make additional fonts available in projects for PLUS+1® graphical terminals.
Uninstall	Displays an **Uninstall** window. Use this window to remove hardware definitions from the **Hardware** tab. Displays an **Uninstall Function Library** window. Use this window to browse to remove function libraries from the **Library** tab. Displays an **Uninstall Font** window. Use this window to view all installed fonts and select fonts to uninstall.
Tools Palette...	Displays an **Icon Menu** window.
Options...	Displays an **Options** window. Use this window to set PLUS+1® GUIDE options such as enabling the **Undo/Redo** commands.

User Interface

Add Menu

The **Add** menu commands add page outlines, buses, ports, text, and other elements to your application.



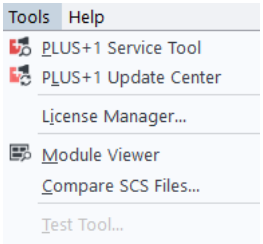
Add menu description

Item	Description
Page	Adds a page where you click and drag in the PLUS+1® GUIDE window's Drawing Area. To edit the top view of the page you just added, click the **Page Interface Editor** command to open the window. For more information, see *Page Interface Editor* on page 103.
Page Interface Editor	Displays the top view page of the current page in a **Page Interface Editor** window. Use the window to edit the top view of the page. For more information, see *Page Interface Editor* on page 103.
Wire Port	Displays the **Select Signal–Schematics Design** window. Use this window to add a Wire Port to where you click in the PLUS+1® GUIDE window's Drawing Area.
Bus Port	Displays a **Select Signal–Schematics Design** window. Use this window to add a Bus Port to where you click in the PLUS+1® GUIDE window's Drawing Area.
Route Wire/Bus	• Starts routing either a green single signal wire or a red multi-signal bus from where you click in the PLUS+1® GUIDE window's Drawing Area. — You get a bus when you start routing from a bus source. — You get a wire when you start routing from a wire source. • K toggles the route between a wire and a bus. • F9 terminates unconnected routes.
Repeat Connection	Duplicates a selected route connection in the PLUS+1® GUIDE window's Drawing Area. Repeated connections automatically have a number added to their name: *Signal_Name, Signal_Name2, Signal_Name3, Signal_Name4...*
Text/Vector Graphics	Displays a menu of text and graphics commands. Use these command to add text and graphic elements to the PLUS+1® GUIDE window's Drawing Area.

User Interface

Tools Menu

The **Tools** menu commands open tools to the service application that you have downloaded, repair errors in your module and manage your PLUS+1® GUIDE license.



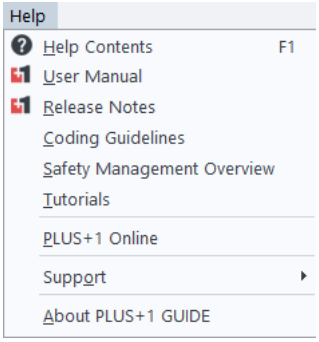
Tools menu description

Item	Description
PLUS+1® Service Tool	Opens the PLUS+1® Service Tool program, Please see *http://www.powersolutions.danfoss.com/products/PLUS-1-GUIDE/GUIDE-service-tool-software-and-license*. Use this tool to: • Download applications to PLUS+1® hardware. • Monitor and change application values. You can open this service program independently of the PLUS+1® GUIDE software.
PLUS+1® Update Center	Opens the PLUS+1® Update Center program.
License Manager	Displays a **Select License** window. Use this window to manage your PLUS+1® GUIDE license. The **user ID** is a number that is unique to your computer. You must send this number to Danfoss Power Solutions to get a license number to unlock the PLUS+1® programs.
Module Viewer	This window provides an additional window, independent of the main PLUS+1® GUIDE window, in which you can edit SCS files.
Compare SCS Files...	Displays a **Compare SCS File** window. Use this window to: • Select the two SCS files whose pages you want to compare for differences. • Output a report on the differences between the pages of the two SCS files that you selected. • Configure the initial display of a **Compare Pages** window, where you explore the differences between the pages in the two SCS files that you selected. For more information, see *Compare SCS Files* on page 554.
Test Tool...	Use the **Test Tool** to perform tests on PLUS+1® GUIDE code.

User Interface

Help Menu

The **Help** menu commands display help and information regarding the PLUS+1® GUIDE software.



Help menu description

Item	Description
Help Contents	Opens a hyperlinked, online Help version of the user manual.
User Manual	Opens a *Portable Document Format* (PDF) version of the user manual, formatted for printing.
Release Notes	Displays *Readme Help* files for the PLUS+1® GUIDE software.
Coding Guidelines	Opens a *Portable Document Format* (PDF) version of the coding guidelines.
Safety Management Overview	Opens a PDF version of the applicable requirements and standards to the PLUS+1® GUIDE software and PLUS+1® Service Tool.
Tutorials	Opens a PDF version of the PLUS+1® Tools tutorials covering practical use cases in GUIDE software and Service tool use cases.
PLUS+1® GUIDE Online	Opens a PLUS+1® GUIDE web page on the Danfoss website. Use this web page as your portal to the latest PLUS+1® GUIDE software downloads and other information about PLUS+1® GUIDE software.
Support	Opens PLUS+1® Support and Training web page on the Danfoss website. This web page has: • Support and training information for PLUS+1® GUIDE software. • Information about what is new in PLUS+1® GUIDE software. • Known issues in PLUS+1® GUIDE software. **Create Troubleshooting File** window creates a ZIP format file of your current application. You can send this file to Danfoss for troubleshooting purposes. **Launch Remote Desktop** window opens a netview support window. Use this window to set up desktop sharing with the Danfoss Help Desk. To complete the setup process, the Help Desk will provide you with a session number to enter into this window.
About PLUS+1® GUIDE	Opens **About PLUS+1 GUIDE** window to see PLUS+1® GUIDE software version and license information. See also: *http://www.powersolutions.danfoss.com/products/PLUS-1-GUIDE/GUIDE-service-tool-software-and-license*

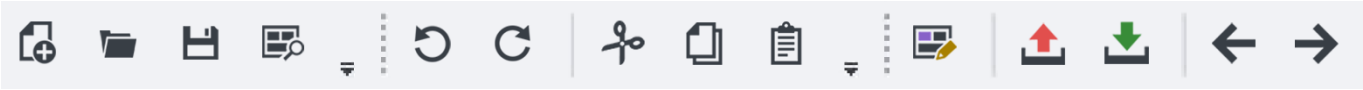
User Interface

Toolbar

Buttons in the **PLUS+1 GUIDE** window toolbar, access commonly used PLUS+1® GUIDE commands.

See *Menus* on page 16 for a complete list of all commands accessed through the menu bar.

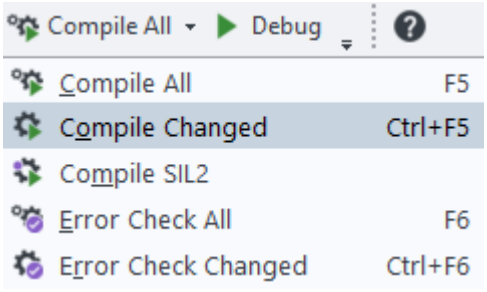
Main toolbar buttons



Main toolbar buttons

Button		Description
	New Project	Displays the **Create New Project** window. Use this window to name a new project and to create a folder for files in your project.
	Open Project	Displays the **Open Project** window. Use this window to locate and open P1P and P1X project files.
	Save Project	Saves the current project files without zipping them into a single P1P file.
	Module Viewer	Displays the **Module Viewer** window. The Module Viewer window provides an additional window, independent of the main GUIDE window, in which you can edit SCS files.
	Undo/Redo	**Undo** — reverses programming actions. **Redo** — reverses Undo actions. Available computer memory determines the number of actions that you can undo. The **Options** window enables and disables the undo/redo function. Click the **Options** button in the toolbar to display this window.
	Copy to Clipboard and Delete Seleted Items	Deletes items selected in the **PLUS+1 GUIDE** window's **Drawing Area** and copies them to the **Clipboard**.
	Copy Selected Objects to Clipboard	Copies items selected in the PLUS+1® GUIDE window's **Drawing Area** to the **Clipboard**.
	Paste from Clipboard	Pastes the contents of the **Clipboard** into the **PLUS+1 GUIDE** window's **Drawing Area**. The link feature automatically links the contents of duplicate pages that are copied from the **Clipboard**.
	Page Interface Editor	Displays the top view page of the current page in the **PLUS+1 GUIDE** window. Use the **Page Interface Editor** window to edit the top view of the page. See *Page Interface Editor* on page 103.
	Leave Page	Leaves the current page.
	Enter Page	Enters a selected page. To enter a page, click within the page boundaries or drag at a page port.
	Previous Page	Moves one page backward through your page navigation history.
	Next Page	Moves one page forward through your page navigation history.

Main toolbar buttons - Compile All, Debug and Help



User Interface

Main toolbar buttons - Compile All, Debug and Helpp

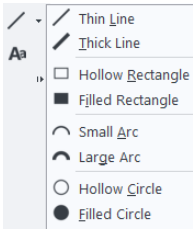
Button		Description
Compile All	**Compile All**	Use the **Compile All** button to select the latest action, or drop the menu to select between compilation and error check actions.
	Compile All	Compiles all modules to produce a downloadable LHX file.
	Compile Changed	Compiles only modules that you have changed since the last Compile All. Use this command to save time when you have more than one module to compile to produce a downloadable LHX file.
	Compile SIL2	Compiles a downloadable LHX file for an application that you certify during the compile process as fulfilling IEC 61508 requirements. For this button to appear in the toolbar, these conditions must be met: • The PLUS+1® program that you are using must both be market-released and certified to comply with IEC 61508 requirements. • The HWD file that you are using must be certified to comply with IEC 61508 requirements. For the application to compile, click **Yes** in a dialog box that opens during the compile process to certify that your application fulfills IEC 61508 requirements. (Certification of a market-released version of the PLUS+1® GUIDE software may occur after its initial release. Contact Danfoss for the certification status of your version of the PLUS+1® GUIDE software.)
	Error Check All	Checks for errors in all modules. Clicking the **Error Check All** or the **Error Check Changed** button does not compile downloadable files.
	Error Check Changed	Compiles to check for errors only in the modules that you have changed since the last Compile All.
Debug	**Debug**	Prepares the application for debugging.
	Help	Displays the complete *PLUS+1® GUIDE Software User Manual*, **AQ152886483724** in PDF format.

Vertical toolbar buttons description

Button		Description
	Quick add component	Quick Search for component to add to **Drawing Area.**
	Route Wire/Bus	Starts routing either a green single signal wire or a red multi-signal bus from where you click in the **Drawing Area**. You get a bus when you start routing from a bus source. You get a wire when you start routing from a wire source. K toggles the route between a wire and a bus. F9 terminates unconnected routes.
	Repeat Connection	Duplicates a selected route connection in the **PLUS+1 GUIDE** window's **Drawing Area**. Repeated connections automatically have a number added to their name: Signal_Name, Signal_Name2, Signal_Name3, Signal_Name4.
	Wire Port	Displays the **Select Signal–Schematics Design** window. Use this window to add a **Wire Port** to where you click in the **Drawing Area**.
	Bus Port	Displays the **Select Signal–Schematics Design** window. Use this window to add a **Bus Port** to where you click in the **Drawing Area**.
	Query/Change Attributes of Selected Objects	Use to change the properties (such as data type, text, and function) of items in the **Drawing Area**. Click an item whose property you want to change. A dialog box appropriate for the selected item displays. Change the property in this window.
	Move Selected Objects to Another Area of Document	Moves items selected in the **PLUS+1 GUIDE** window's **Drawing Area**.
	Stretch	Stretches segments in selected routes and moves selected items. Click a route to add a vertex to the route. The new vertex can then be selected and moved without changing the positions of other vertexes in the route.

User Interface

Vertical toolbar buttons description (continued)

Button		Description
	Copy Selected Objects to Another Area of Document	Copies items selected in the **PLUS+1 GUIDE** window's **Drawing Area**. Selected items turn white. Click to place copied items. Press Esc to stop placing copied items. This command only works within the **Drawing Area**. The link feature automatically links copied pages.
	Delete Selected Objects	Deletes items selected in the **PLUS+1 GUIDE** window's **Drawing Area**. Selected items turn white. The **Attributes** window displays when you select a single item or identical items. In the **Attributes** window, click **OK** to delete your selection. The **Select Item Class** window displays when you select different items. Use this window to select the items that you want to delete: • Asterisks (*) identify items that will be deleted. • Dashes (–) identify items that will not be deleted. In the **Select Item Class** window, click **OK** to delete your selections.
	**Thin Line** - Graphical tools menu	Displays a menu of graphics commands. Use this command to add vector graphic elements to the Drawing Area.
	Text Box	Displays a Rich text box editor. Use this command to add text and images to the Drawing Area.

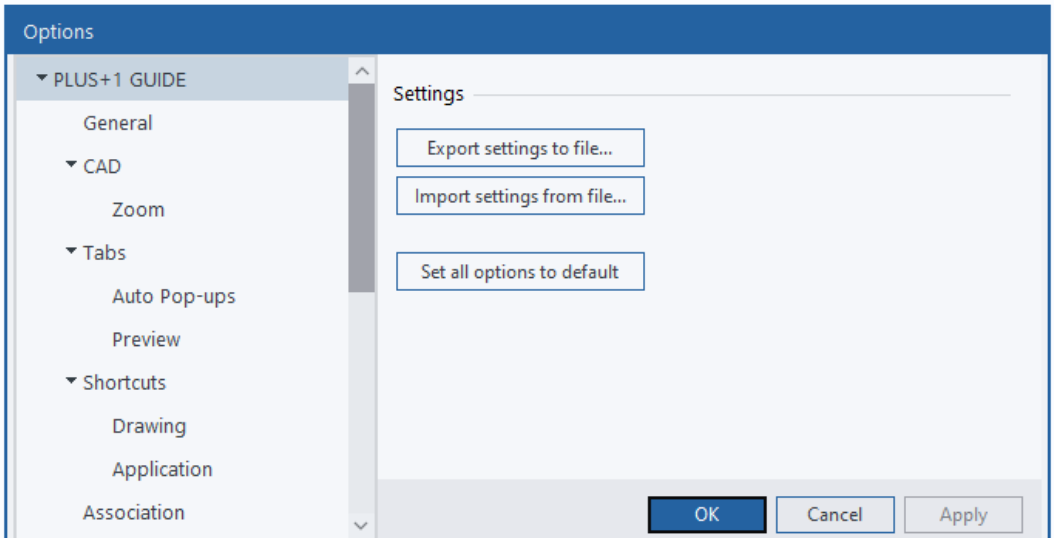
User Interface

Dialogs

Options

This chapter describes the configuration settings available in the **Options** dialog window.

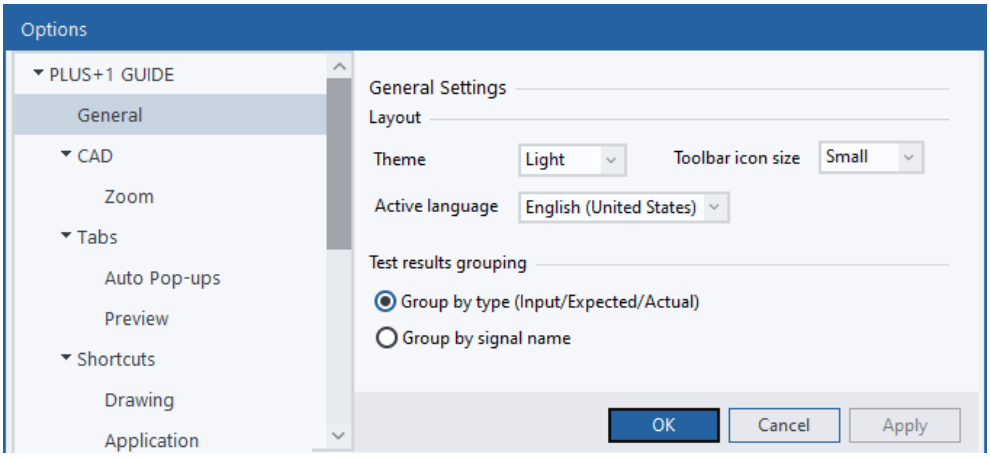
PLUS+1 GUIDE Settings



PLUS+1 GUIDE settings description

Item	Description
Export settings to file	Export all your GUIDE settings to a file.
Import settings from file	Overwrites all your GUIDE settings with settings from a file.
Set all options to default	Overwrites all your GUIDE settings to their default values.

General Settings

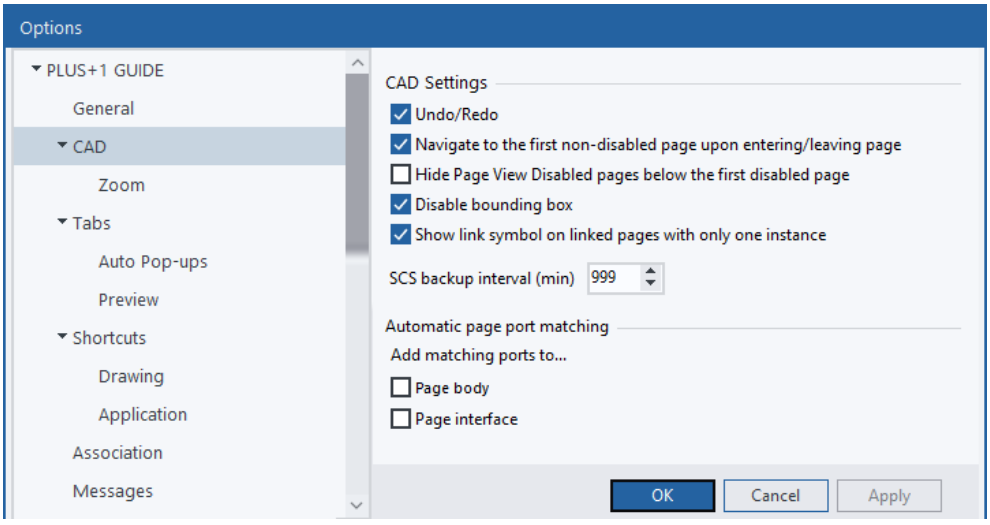


User Interface

General Settings

Item	Description
Layout	
Theme	Switch between light and dark themes.
Toolbar icon size	Switch between small and large toolbar icons.
Active language	Switch between English and Chinese.
Test results grouping	
Group by type (Input/Expected/Actual)	Switch between displaying test result columns by type.
Group by signal name	Switch between displaying test result columns by name.

CAD Settings



CAD Settings

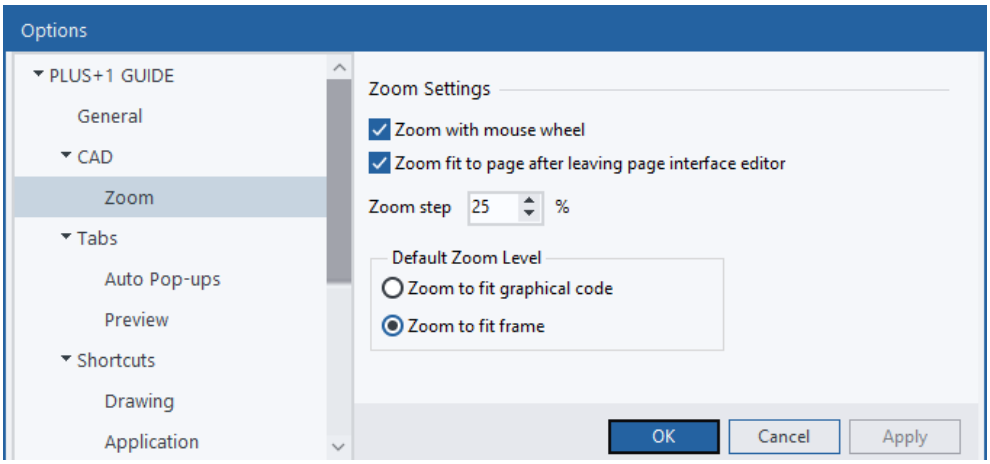
Item	Description
Undo/Redo	When checked, enables the **Undo** and **Redo** commands.
Navigate to the first non-disabled page upon entering/leaving page	When checked, entering or leaving a page with a **Page View Access** property of **Disabled**, navigates to the first page having a **Page View Access** property of **Normal**, **Force Enabled**, or **Read-only**.
Hide Page View Disabled pages below the first disabled page	• When checked, the **Page Navigator** tab hides pages that are inside the top page if both the top page and the inside pages have the **Page View Access** property of **Disabled**. • When unchecked, the **Page Navigator** tab dims out pages that have a **Page View Access** property of **Disabled**.
Disable bounding box	When checked, the bounding box of the drawing is disabled, and hence not shown.
Show link symbol on linked pages with only one instance	When checked, the link symbol will be displayed on all linked pages, even linked pages that only have one instance. Default is unchecked, which corresponds to the behavior in versions before 2022.1.
SCS backup interval	Sets how often the PLUS+1® GUIDE software automatically backs up the SCS file. Making a change in the **Drawing Area** starts the countdown to the next backup. The PLUS+1® GUIDE software saves backup files (~*.scs) to the current project folder. **Range: 1–999 min**.
Automatic page port matching	

User Interface

CAD Settings (continued)

Item	Description
Add matching ports to...	
Page body	When checked, a port added to the **Page interface** view (top view) of a page duplicates itself within the page. Ports added in the **Page interface** editor window appear in the upper left-hand corner of the page.
Page interface	When checked, a port added to a page duplicates itself in the **Page interface** view (top view) of the page. Ports added to a page appear in the upper left-hand corner of the **Page interface** editor window.

Zoom Settings

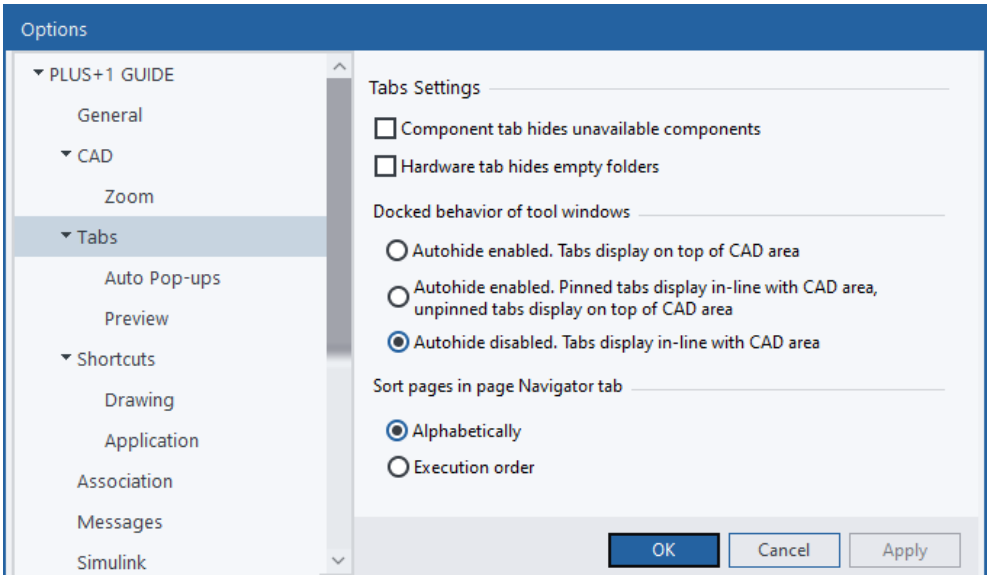


Zoom Settings

Item	Description
Zoom with mouse wheel	• When checked, the mouse wheel zooms the **Drawing Area**. • When unchecked, the mouse wheel does not zoom the **Drawing Area**.
Zoom fit to page	• When checked, the view homes when you close the **Page Interface Editor** window. (This works as if you pressed **HOME** after closing the **Page Interface Editor** window.) • When unchecked, the view does not home when you close the **Page Interface Editor** window.
Zoom Step	Sets the percentage change in magnification with each: • Notch of the mouse wheel. • Press of `PageUp` or `PageDown` button.
Default Zoom Level	• **Zoom to fit graphical code:** The same zoom setting as in previous versions of PLUS +1® GUIDE. All code on a page will be visible, but not necessarily the frame. • **Zoom to fit frame:** This gives a consistent zoom level for every page, regardless of the amount of code. Code drawn outside of the frame will not be immediately visible.

User Interface

Tabs Settings

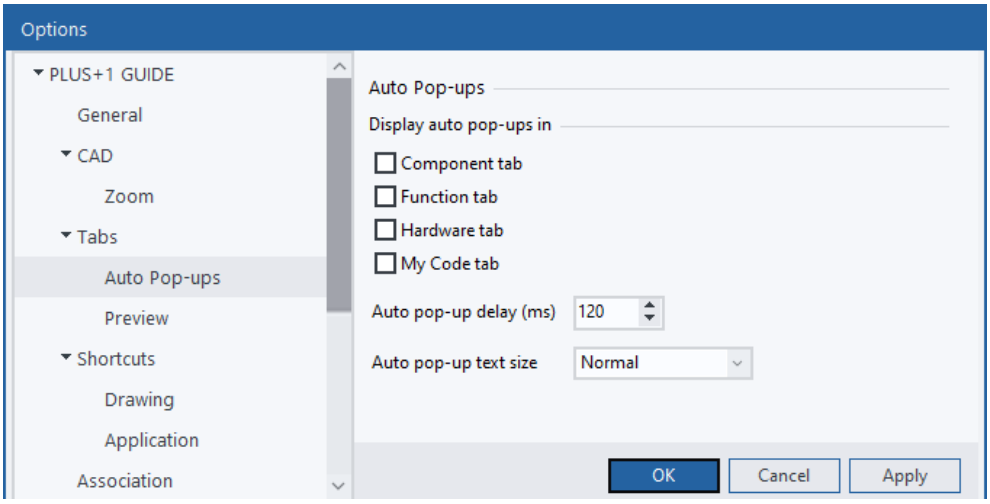


Tabs Settings

Item	Description
Component tab hides unavailable components	When checked, the **Component** tab hides the components that cannot be used with the selected hardware.
Hardware tab hides empty folders	When checked, the **Hardware** tab hides empty folders in its hardware tree.
Docked behavior of tool windows	
Autohide enabled. Tabs display on top of CAD area	When selected, the tool windows will display on top of the CAD area. This is the best mode for users that use autohide on all tabs and don't pin them.
Autohide enabled. Pinned tabs display in-line with CAD area, unpinned tabs display on top of CAD area	Hybrid mode. This is the best mode for users that want autohide on some tabs and pin others.
Autohide disabled. Tabs display in-line with CAD area	When selected, the CAD area and tool windows will be displayed side-by-side. Auto-hiding of tool windows will not be available in this mode. This is the best mode for users that don't use the autohide function.
Sort pages in page Navigator tab	
Alphabetically	When selected, the pages in the page navigator tree will be sorted alphabetically.
Execution order	When selected, the pages in the page navigator tree will be sorted by program, execution order.

User Interface

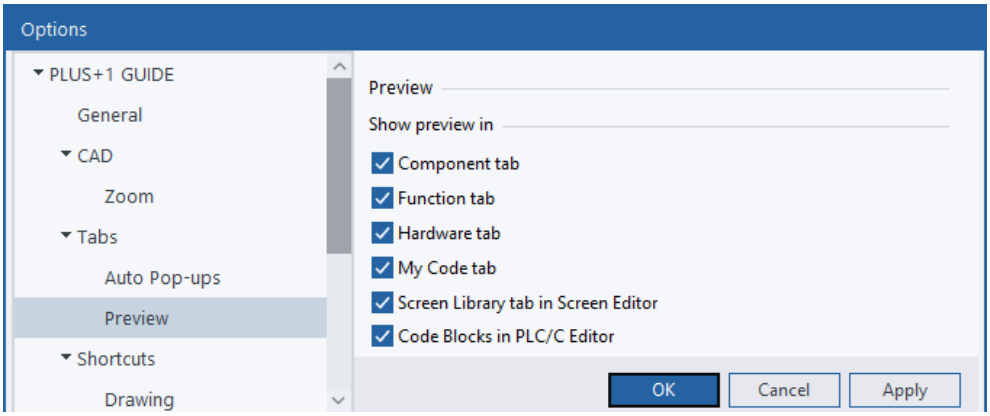
Auto Pop-Ups Settings



Auto Pop-Ups Settings

Item	Description
Display auto pop-ups in	
Component tab	When checked, hovering over an unexpanded branch in the corresponding tree displays an auto pop-up menu that shows all the items within the branch.
Function tab	
Hardware tab	
My Code tab	
Auto pop-up delay (ms)	Sets the delay before a tab displays a pop-up menu for the branch you click. **Range: 10–5000 ms**
Auto pop-up text size	Select one of: • **Normal** — pop-up menus display icons. • **Small** (No icons) — pop-up menus use a smaller text and do not display icons.

Preview Settings



User Interface

Preview Settings table

Item	Description
Component tab	When checked, the preview functionality will be enabled in this tab.
Function tab	When checked, the preview functionality will be enabled in this tab.
Hardware tab	When checked, the preview functionality will be enabled in this tab.
My Code tab	When checked, the preview functionality will be enabled in this tab.
Screen Library tab in Screen Editor	When checked, the preview functionality will be enabled in this tab.
Code Blocks in PLC/C Editor	When checked, the preview functionality will be enabled in this tab.

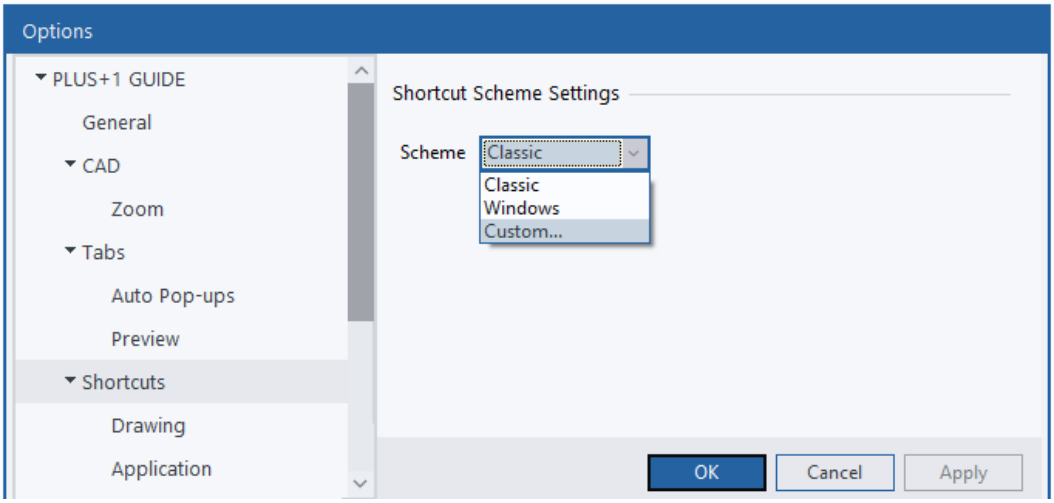
User Interface

Shortcut Scheme Settings

Shortcut Scheme Settings enable **Classic**, **Windows** or **Custom** shortcut key schemes by Command and Shortcut key combinations.

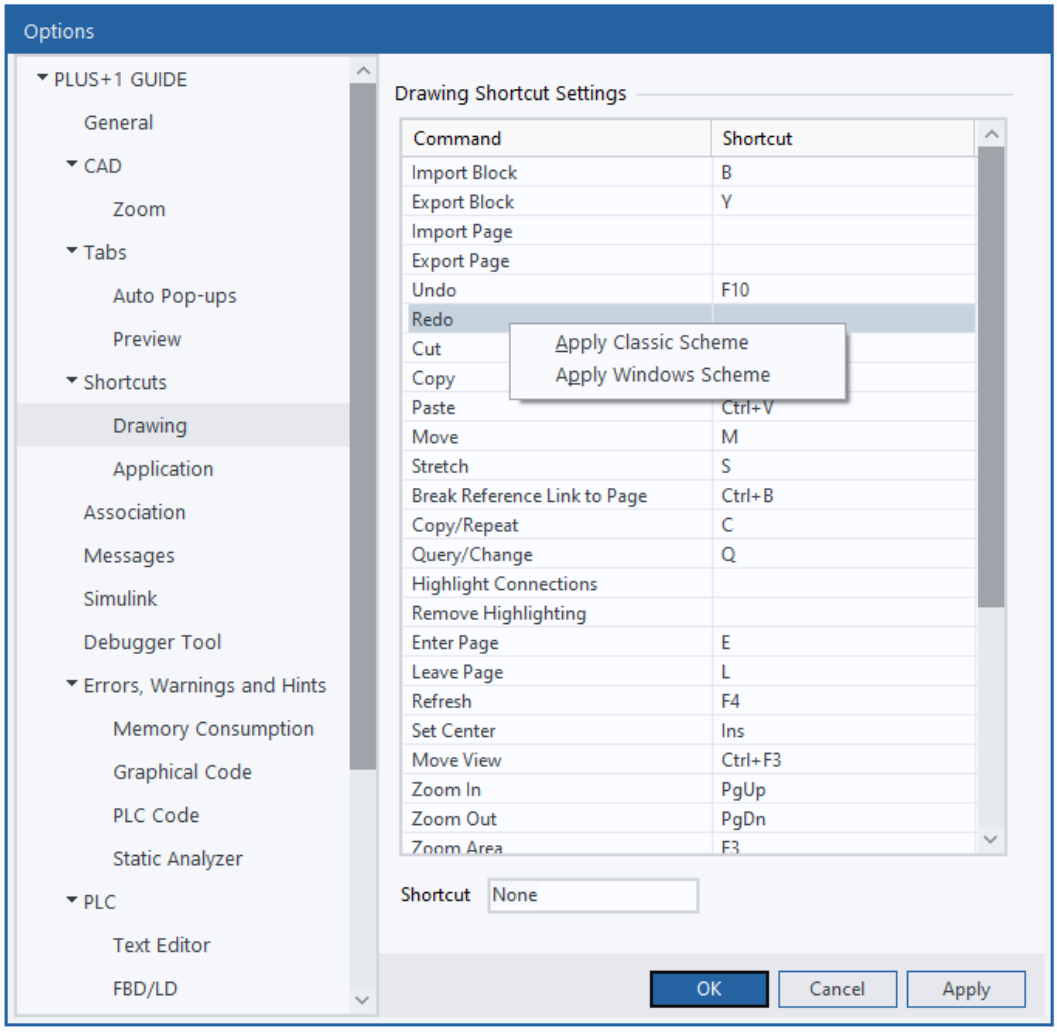
Classic and **Windows** shortcut schemes cannot be changed.

1. In the **Scheme** drop-down list, select **Custom** to edit the custom scheme.



User Interface

2. Under the **Shortcuts** tab, select **Drawing** or **Application** to choose settings.

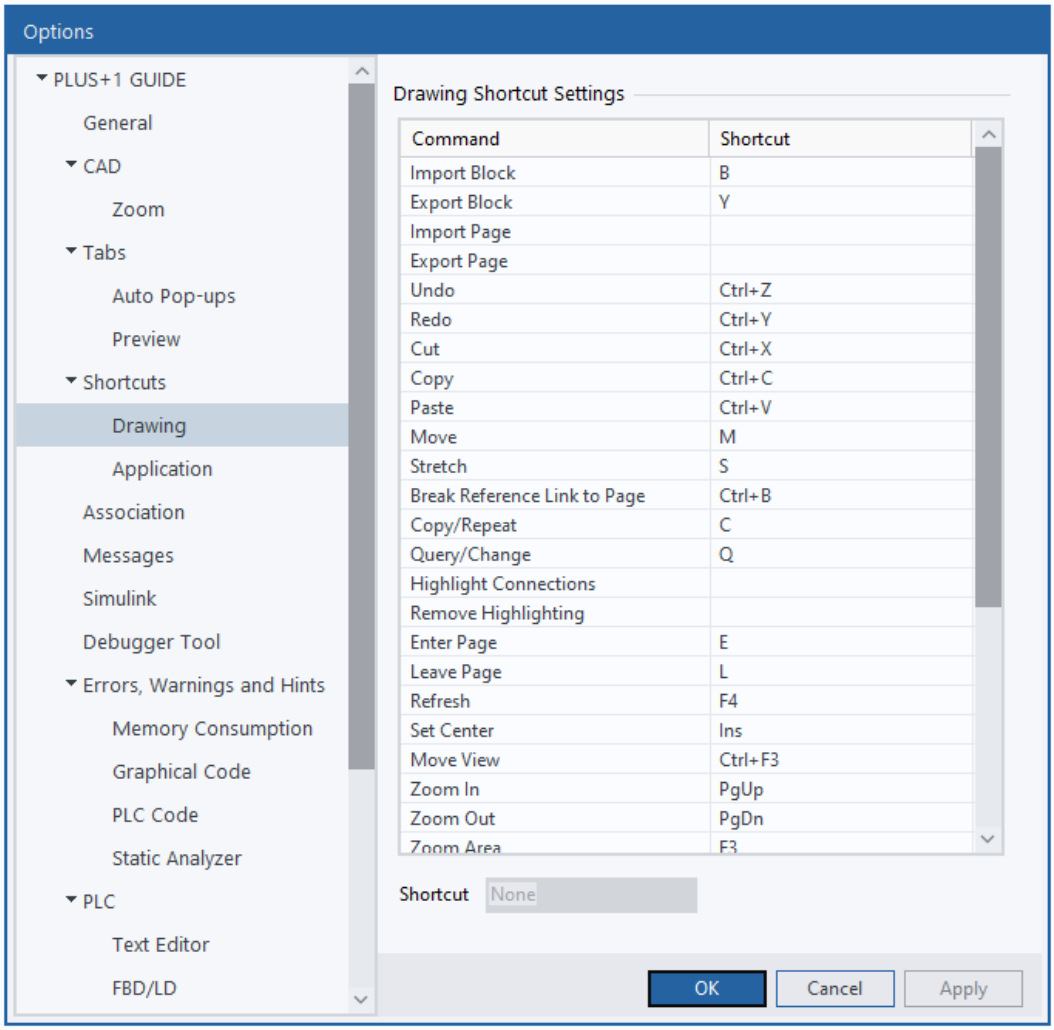


3. Right-click on a **Command** in the **Shortcuts Settings** field to display a pop-up menu.
4. In the pop-up menu, select **Apply Classic Scheme** or **Apply Windows Scheme** to select a starting scheme for the custom shortcuts.
5. In the **Shortcut** field, enter the desired shortcut.

User Interface

Classic and Windows Shortcuts

Classic and Windows Drawing Shortcut Key Schemes



Classic and Windows Drawing Shortcut Key Schemes

Command	Classic scheme shortcuts	Windows scheme shortcuts
Import Block	**B**	**B**
Export Block	**Y**	**Y**
Import Page	No default shortcut set	No default shortcut set
Export Page		
Undo	**F10**	**Ctrl+Z**
Redo	No default shortcut set	**Ctrl+Y**
Cut	**Ctrl+X**	**Ctrl+X**
Copy	**Ctrl+C**	**Ctrl+C**
Paste	**Ctrl+V**	**Ctrl+V**
Move	**M**	**M**
Stretch	**S**	**S**

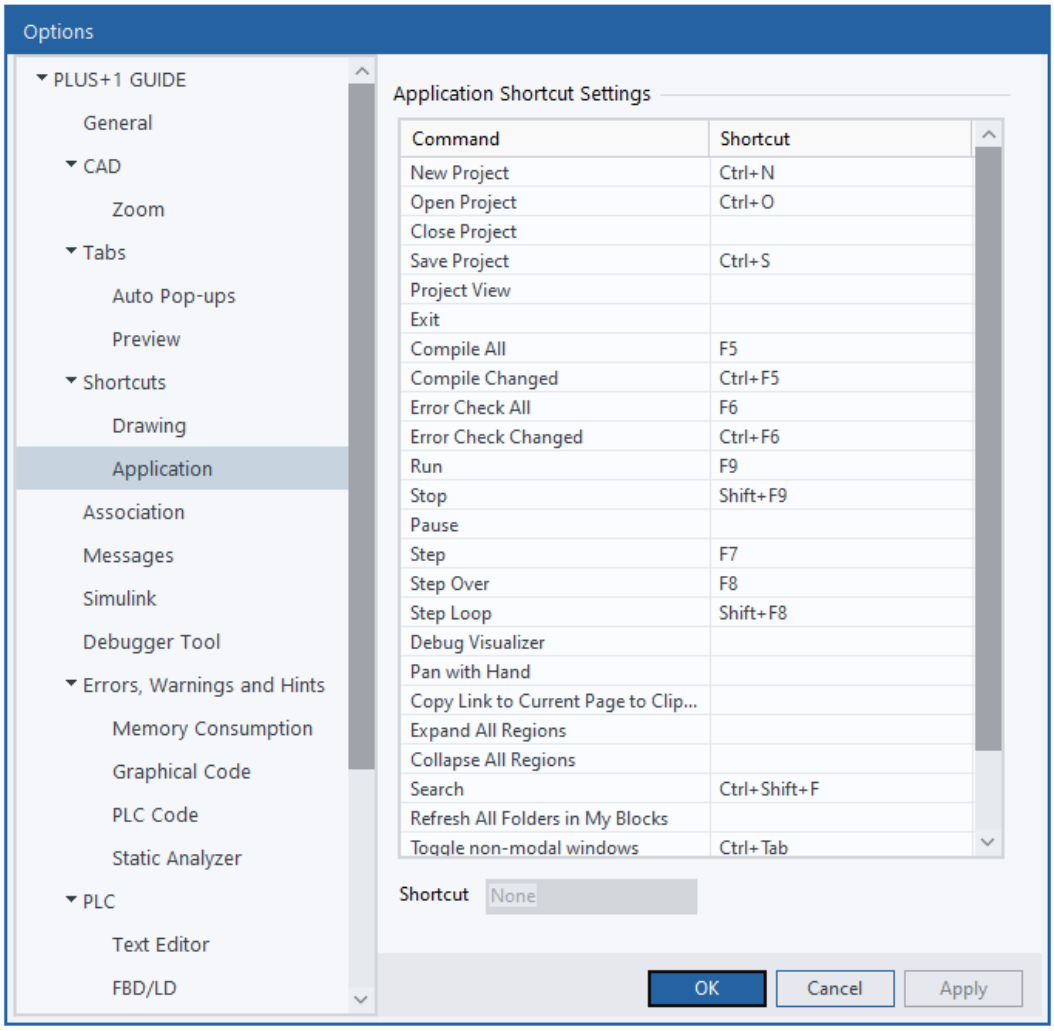
User Interface

Classic and Windows Drawing Shortcut Key Schemes (continued)

Command	Classic scheme shortcuts	Windows scheme shortcuts
Break Reference Link to Page	**Ctrl+B**	**Ctrl+B**
Copy/Repeat	**C**	**C**
Query/Change	**Q**	**Q**
Highlight Connections	No default shortcut set	No default shortcut set
Remove Highlighting		
Enter Page	**E**	**E**
Leave Page	**L**	**L**
Refresh	**F4**	**F4**
Set Center	**Ins**	**Ins**
Move View	**Ctrl+F3**	**Ctrl+F3**
Zoom In	**PgUp**	**PgUp**
Zoom Out	**PgDn**	**PgDn**
Zoom Area	**F3**	**F3**
Zoom Fit Page	**Home**	**Home**
Toggle Display Grid	**G**	**G**
Route Wire/Bus	**R**	**R**
View Logical Net	**V**	**V**
Add Text	No default shortcut set	No default shortcut set
Transfer Between Pages		
Repeat Connection		
View Dependencies		
Add Text Box		

Classic and Windows Application Scheme Shortcuts

User Interface



Classic and Windows Application Scheme Shortcuts

Command	Classic and Windows Key Shortcuts
New Project	**Ctrl+N**
Open Project	**Ctrl+O**
Close Project	No default shortcut set
Save Project	**Ctrl+S**
Project View	No default shortcut set
Exit	
Compile All	**F5**
Compile Changed	**Ctrl+F5**
Error Check All	**F6**
Error Check Changed	**Ctrl+F6**
Run	**F9**
Stop	**Shift+F9**
Pause	No default shortcut set

User Interface

Classic and Windows Application Scheme Shortcuts (continued)

Command	Classic and Windows Key Shortcuts
Step	**F7**
Step Over	**F8**
Step Loop	**Shift+F8**
Pan with Hand	No default shortcut set
Copy Link to Current Page to Clipboard	
Expand All Regions	
Collapse All Regions	
Search	**Ctrl+Shift+F**
Refresh All Folders in My Blocks	No default shortcut set
Toggle non-modal windows	**Ctrl+Tab**
Use the default language	**Ctrl+Alt+F**
Quick add component	**Ctrl+Shift+A**

User Interface

Drawing Sub Commands

Some drawing commands temporarily enable sub commands while active. The following table lists those sub commands.

Mirrored or rotated component is not recommended because the execution order will be affected.

- Only mirror ports.
- Only rotate Text/Vector Graphics elements.

Sub commands

Main command	Condition	Sub command	Description
Move	A component has been selected for move	**F8**	Mirror the component.
		F9	Rotate the component by 90 degrees.
		F7	Rotate the component by 45 degrees.
		F6	Rotate the component by 1 degree.
Route	Routing has started	**F9**	Complete the routing action.
		k	Switch between routing wire/bus.
		s	Switch between free or forced orthogonal routing.

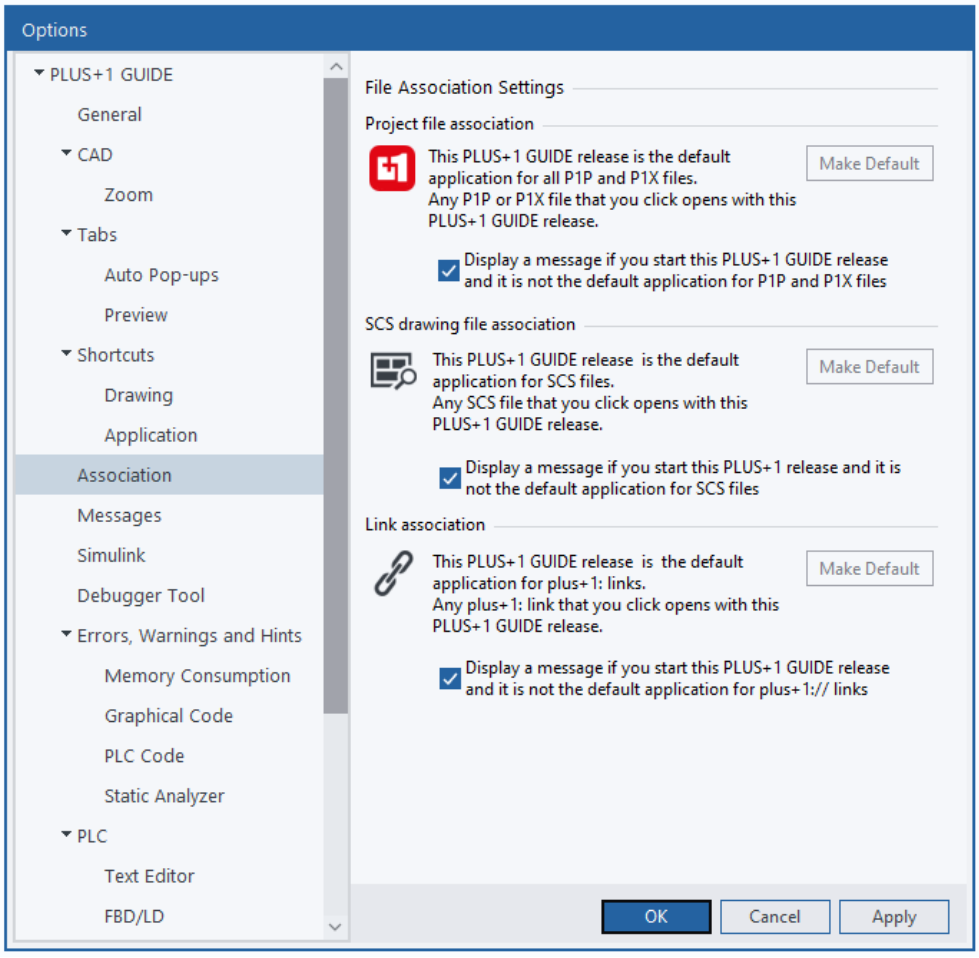
ESC will cancel the current drawing command.

User Interface

File Association Settings

File Association Settings helps manage running different releases of PLUS+1® GUIDE on the same PC.

- PLUS+1® GUIDE releases 3.3 and later support running different releases installed on the same PC.
- Only one instance of PLUS+1® GUIDE can run at a time.



Options Window—Project File Association Settings

Item	Description
Make Default	Check to make this PLUS+1® GUIDE release the default application for all P1X and P1P files. Clicking a file with a P1X or P1P extension automatically opens the file with this PLUS +1® GUIDE release.
Check at startup	Check to display a message if you open this PLUS+1 GUIDE release and it is not the default application for P1X and P1P files.

Options Window—SCS Drawing File Association Settings

Item	Description
Make Default	Check to make this PLUS+1® GUIDE release the default application for all SCS files. Clicking a file with an SCS extension automatically opens the file with this PLUS+1® GUIDE release.
Check at startup	Check to display a message if you open this PLUS+1® GUIDE release and it is not the default application for SCS files.

User Interface

Options Window—Link Association Settings

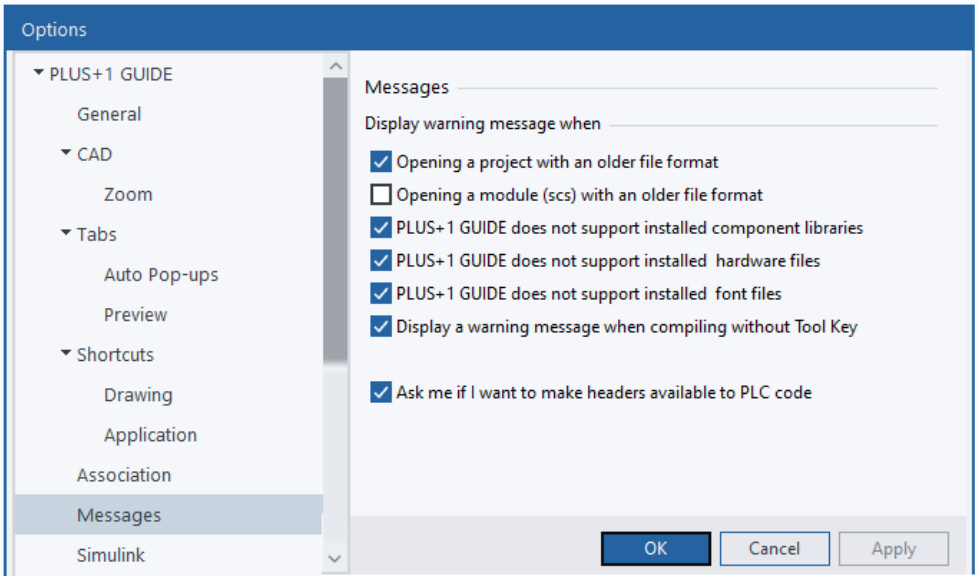
Item	Description
Make Default	Check to make this PLUS+1® GUIDE release the default application for all plus+1 links. Clicking a plus+1 link automatically opens the file with this PLUS+1® GUIDE release.
Check at startup	Check to display a message if you open this PLUS+1® GUIDE release and it is not the default application for plus+1 links.

User Interface

Messages

Messages manage the display of messages that appear when you run different versions of the PLUS+1® GUIDE software on the same PC.

- PLUS+1® GUIDE software releases 3.3 and later support running different releases installed on the same PC.
- Only one PLUS+1® GUIDE software can run at time.

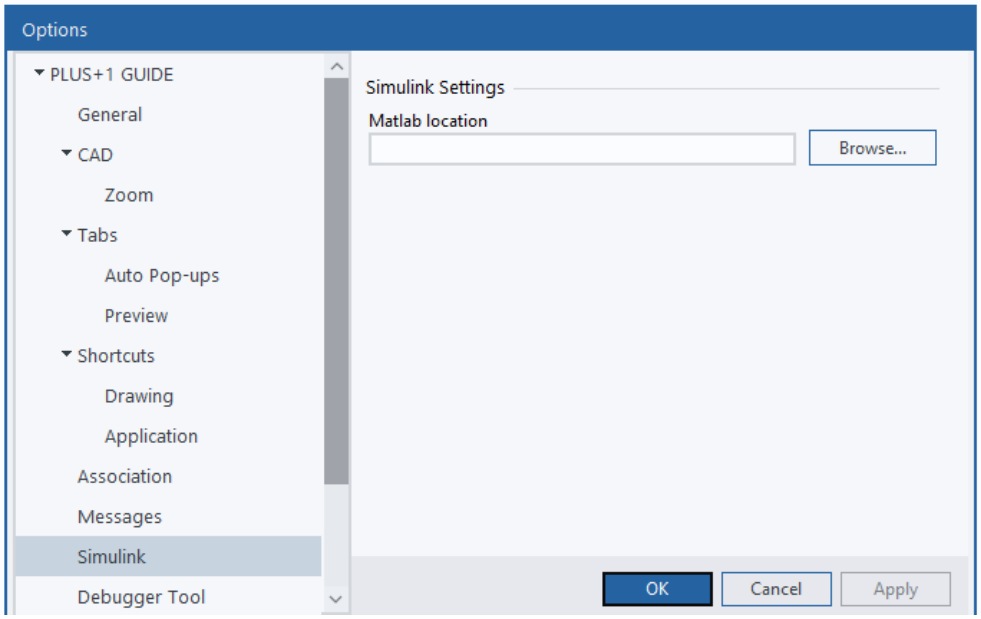


Options Window—Messages description

Item	Description
Display warning message when	
Opening a project with an older file format	Check to display a warning message if your PLUS+1® GUIDE release opens a project with an older file format.
Opening a module (scs) with an older file format	Check to display a warning if your PLUS+1® GUIDE release opens a module (SCS file) with an older file format.
PLUS+1 GUIDE does not support installed component libraries	Check to display a warning message if your PLUS+1® GUIDE release does not support installed component libraries.
PLUS+1 GUIDE does not support installed harware files	Check to display a warning message if your PLUS+1® GUIDE release does not support installed hardware files.
PLUS+1 GUIDE does not support installed font files	Check to display a warning message if your PLUS+1® GUIDE release does not support installed font files.
Display a warning message when compiling without Tool Key	Check to display a warning message when you compile an application without the Tool Key security feature. Tool Key protection reduces the risk that unauthorized personnel could use the PLUS+1® Service Tool application to view and change your application's operating parameters.
Ask me if I want to make headers available to PLC code	In a project that contains C header files, none of which have been made available to PLC code, GUIDE will ask the user to add one or several of these if this option is checked.

User Interface

Simulink Settings



Simulink Settings defines the path to the executable file for the Simulink program.

This is a PLUS+1® GUIDE upgrade feature. A GUIDE-to-Simulink License (PLUS+1® GUIDE add on license Simulink Data Sheet, **AI170686484195**) enables this feature. For more information, see PLUS+1® GUIDE Licensing, **AQ419969404812**.

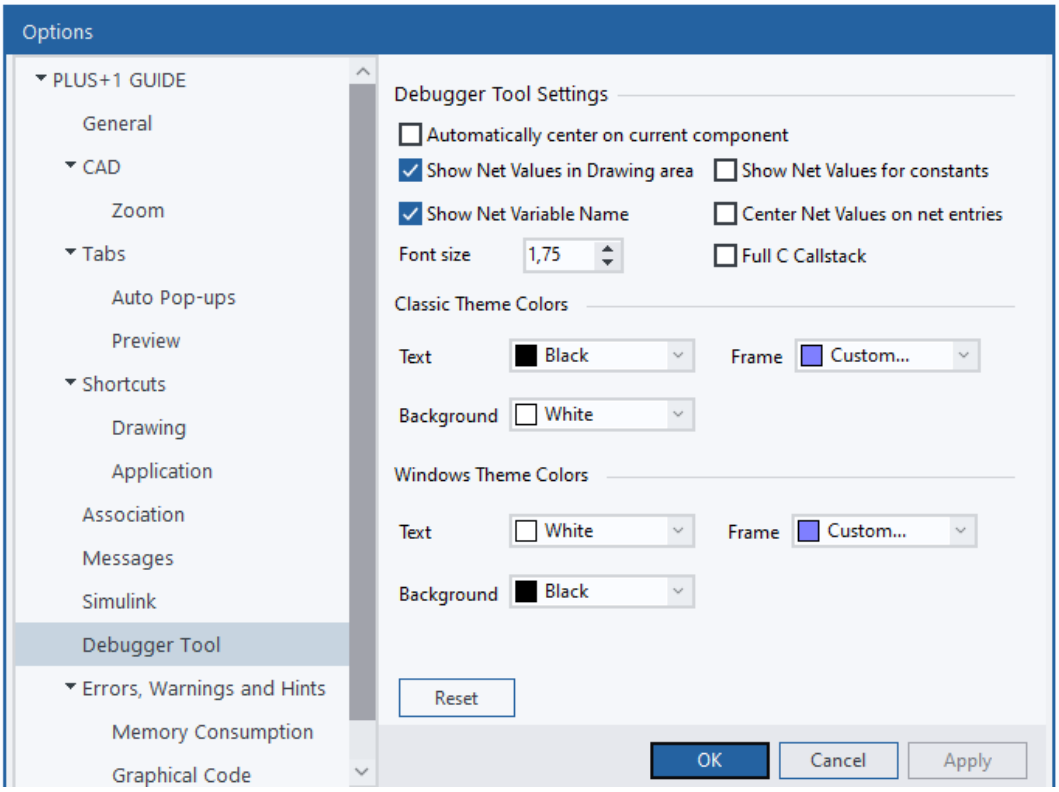
Options Window—Simulink Settings description

Item	Description
Matlab location	Sets the path to the executable file for the Simulink program.

User Interface

Debugger Tool Settings

Debugger Tool Settings customize the performance and appearance of the Debugger Tool.

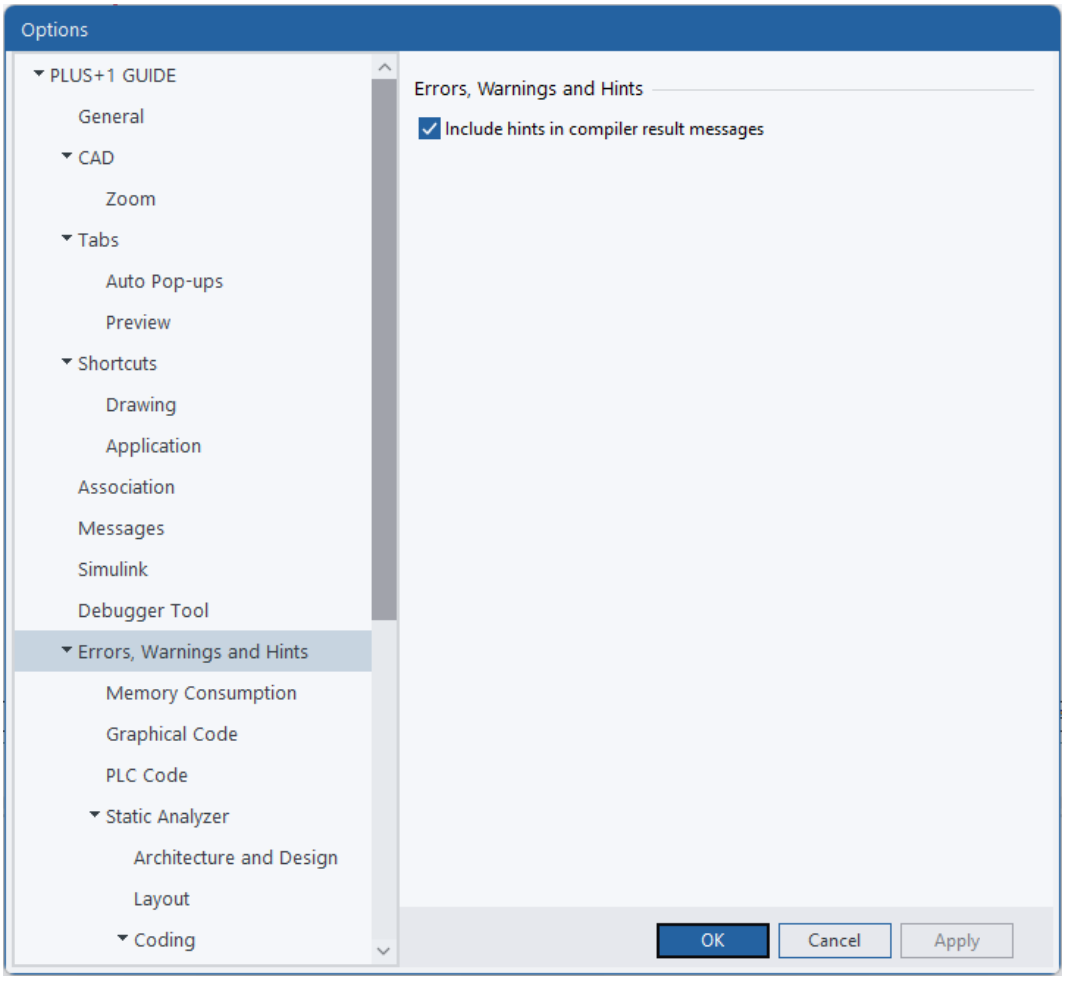


Debugger Tool settings description

Item	Description
Automatically center on current component	When checked - as you step through components the **Drawing Area** shifts to center on the current component.
Show Net Values in Drawing area	Shows **Net Values** in the **Drawing Area**.
Show Net Variable Name	Shows **Net Variable Name** in the **Drawing Area**.
Show Net Values for Constants	Shows **Net Values** for constant nets.
Center Net Values on net entries	• When checked, centers **Net Values** on the inputs and outputs of components. • When unchecked, offsets **Net Values** from the inputs and outputs of components.
Full C Callstack	Displays the full C callstack when debugging. This will show both functions generated from user code (or direct user code in the case of C functions), as well as functions generated completely by the tool.
Font size	Sets the size of the **Net Values** text.
Classic Theme Colors	Customizes the appearance of **Net Values** when your **Color Scheme** choice is **Classic**. • **Text**—sets the color of **Net Values** text. • **Background**—sets the background color for **Net Values**. • **Frame**—sets the color of the frame that surrounds **Net Values**.
Windows Theme Colors	Customizes the appearance of Net Values when your **Color Scheme** choice is **White**. • **Text**—sets the color of **Net Values** text. • **Background**—sets the background color for **Net Values**. • **Frame**—sets the color of the frame that surrounds **Net Values**.

User Interface

Errors, Warnings and Hints Settings

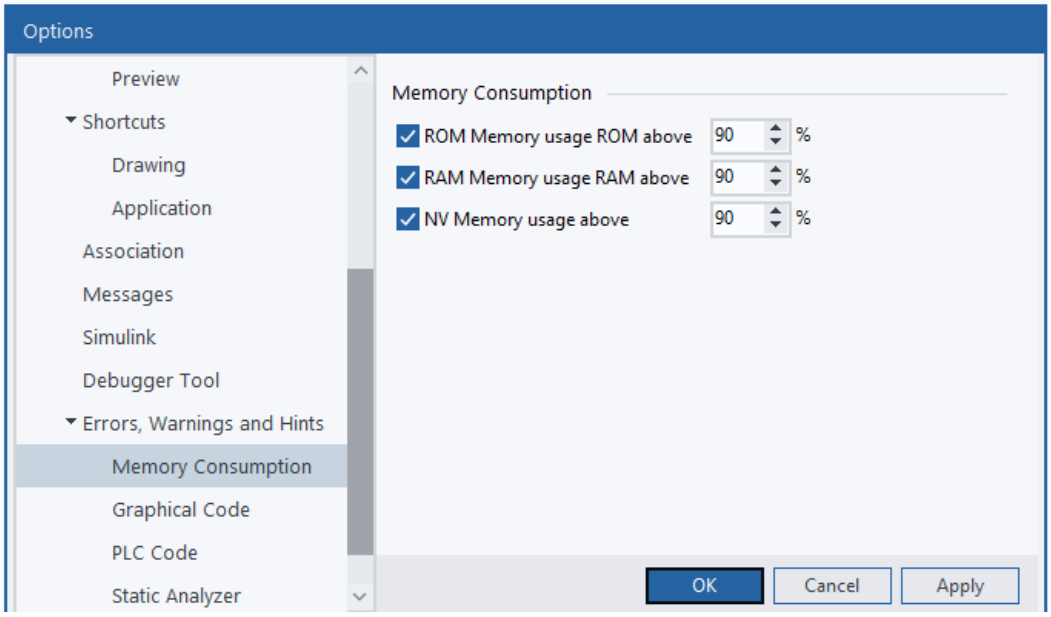


Errors, Warnings and Hints settings description

Item	Description
Include hints in compiler result messages	• When checked, hints will be included in the **Errors, Warnings and Hints** tab • When unchecked, hints will not be included in the **Errors, Warnings and Hints** tab

User Interface

Memory Consumption Settings

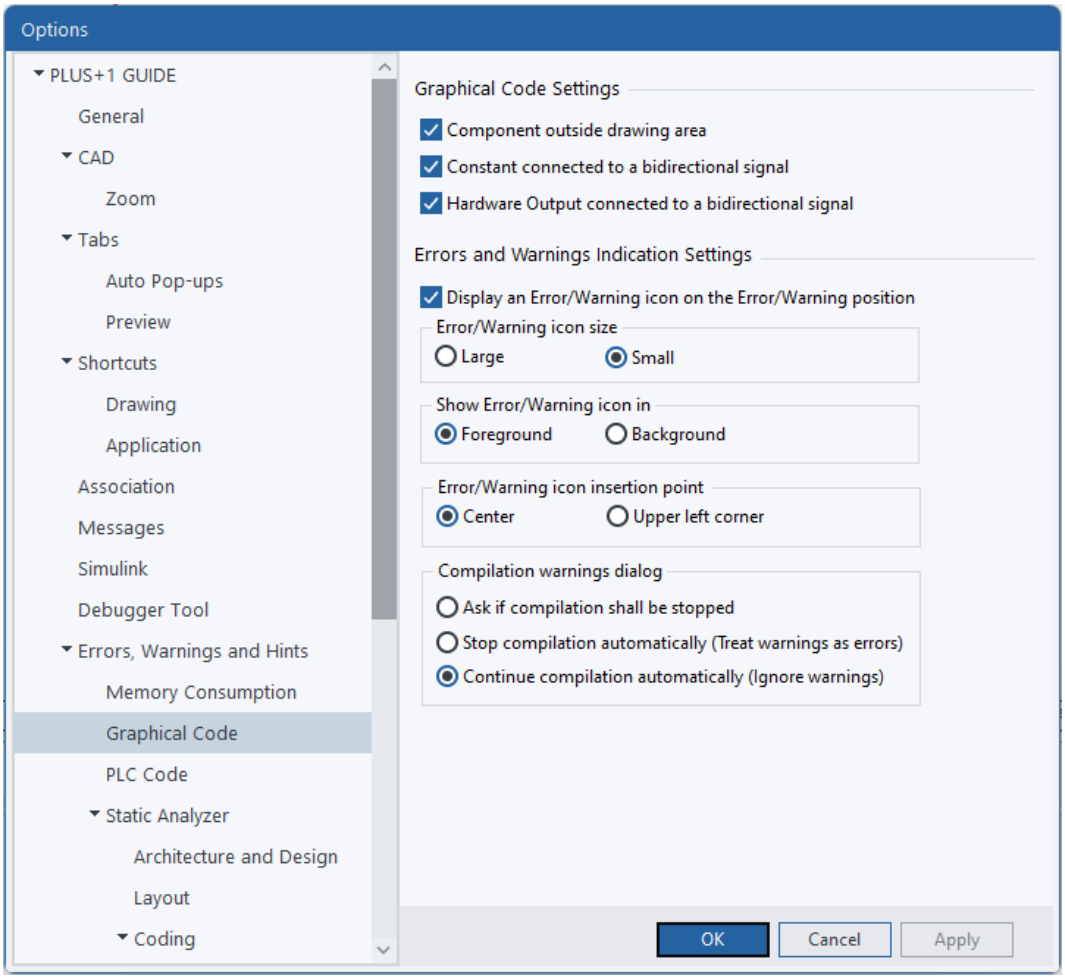


Memory Consumption settings description

Item	Description
ROM Memory usage ROM	Threshold value on percentage ROM Memory used to report warning. Enables ROM Memory check.
RAM Memory usage RAM	Threshold value on percentage RAM Memory used to report warning. Enables RAM Memory check.
NV Memory usage	Threshold value on percentage NV Memory used to report warning. Enables NV Memory check.

User Interface

Graphical Code Settings



Graphical Code settings description

Item	Description
Component outside drawing area	Enables warning on component outside drawing area.
Constant connected to a bidirectional signal	Enables warning on constant connected to bidirectional signal.
Hardware Output connected to a bidirectional signal	Enables warning on HW output connected to bidirectional signal.
Display an Error/Warning icon on the Error/Warning position	• When checked, clicking an error or warning message in the **Error Messages** tab displays an **Error/Warning** icon at the insertion point (page x-y coordinates) where the compiler found the error or warning condition. • When unchecked , clicking an error or warning message in the **Error/Warning/Hint Messages** tab does not display an **Error/Warning** icon at the insertion point (page x-y coordinates) where the compiler found the error or warning condition.
Error/Warning icon size	• **Large** — display large Error/Warning icons. • **Small** — display small Error/Warning icons.
Show Error/Warning icon in	• **Foreground** — display the Error/Warning icon in front of the drawing. • **Background** — display the Error/Warning icon behind the drawing.

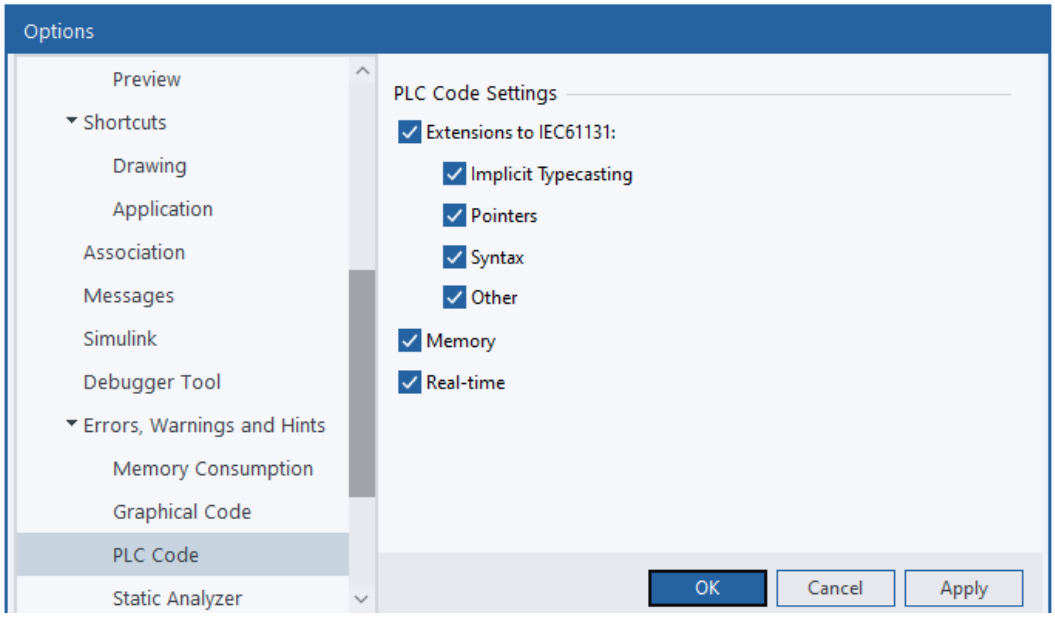
User Interface

Graphical Code settings description (continued)

Item	Description
Error/Warning icon insertion point	• **Center** — centers the Error/Warning icon on insertion point (page x-y coordinates) where the compiler found the error or warning condition. • **Upper left corner** — offsets the Error/Warning icon. The insertion point (page x-y coordinates) where the compiler found the error or warning condition is above and to the left of the Error/Warning icon.
Compilation warnings dialog	• **Ask if compilation shall be stopped** — displays a warning message with an option to continue compiling. • **Stop compilation automatically (Treat warnings as errors)** — displays a warning message and terminates the compile process. • **Continue compilation automatically (Ignore warnings)** — displays no warning messages during the compile process.

User Interface

PLC Code Settings



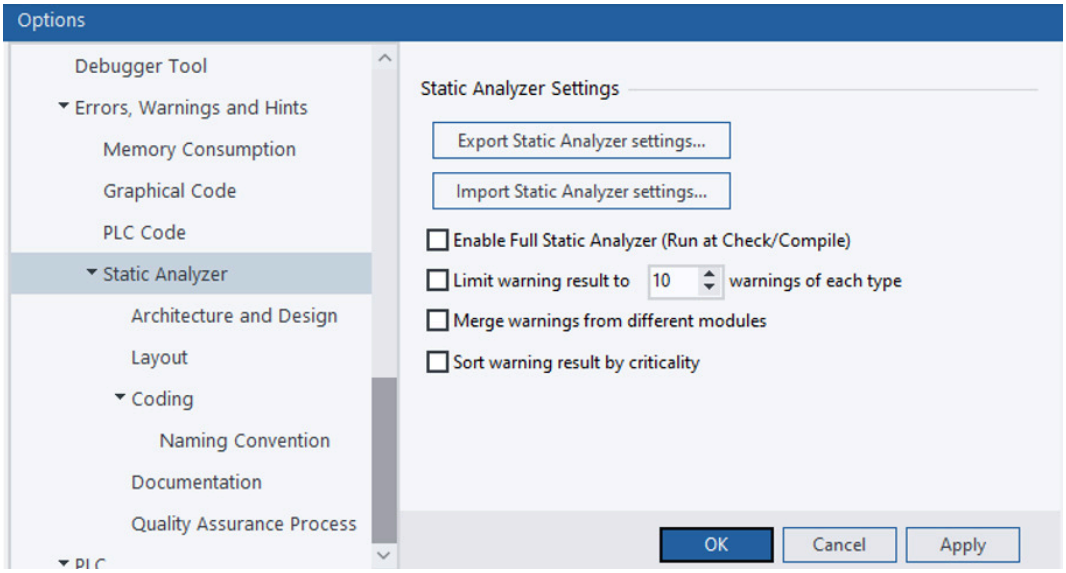
PLC Code settings description

Item	Description
Extensions to IEC61131	When checked, enables all warnings on extensions to IEC61131. When unchecked, allows detailed settings to enable/disable each group of warnings.
Implicit Typecasting	Enables warnings related to implicit typecasting. (Enabled indirectly by enabling Extensions to IEC61131.)
Pointers	Enables warnings related to usage of pointers. (Enabled indirectly by enabling Extensions to IEC61131.)
Syntax	Enables warnings related to issues on syntax. (Enabled indirectly by enabling Extensions to IEC61131.)
Other	Enables other warnings related to IEC61131 extensions. (Enabled indirectly by enabling Extensions to IEC61131.)
Memory	Enables warnings related to memory issues.
Real-time	Enables warning related to real-time behavior.

User Interface

Static Analyzer Settings

The descriptions for the **Static Analyzer** settings are listed also with the warning codes for detected warnings.

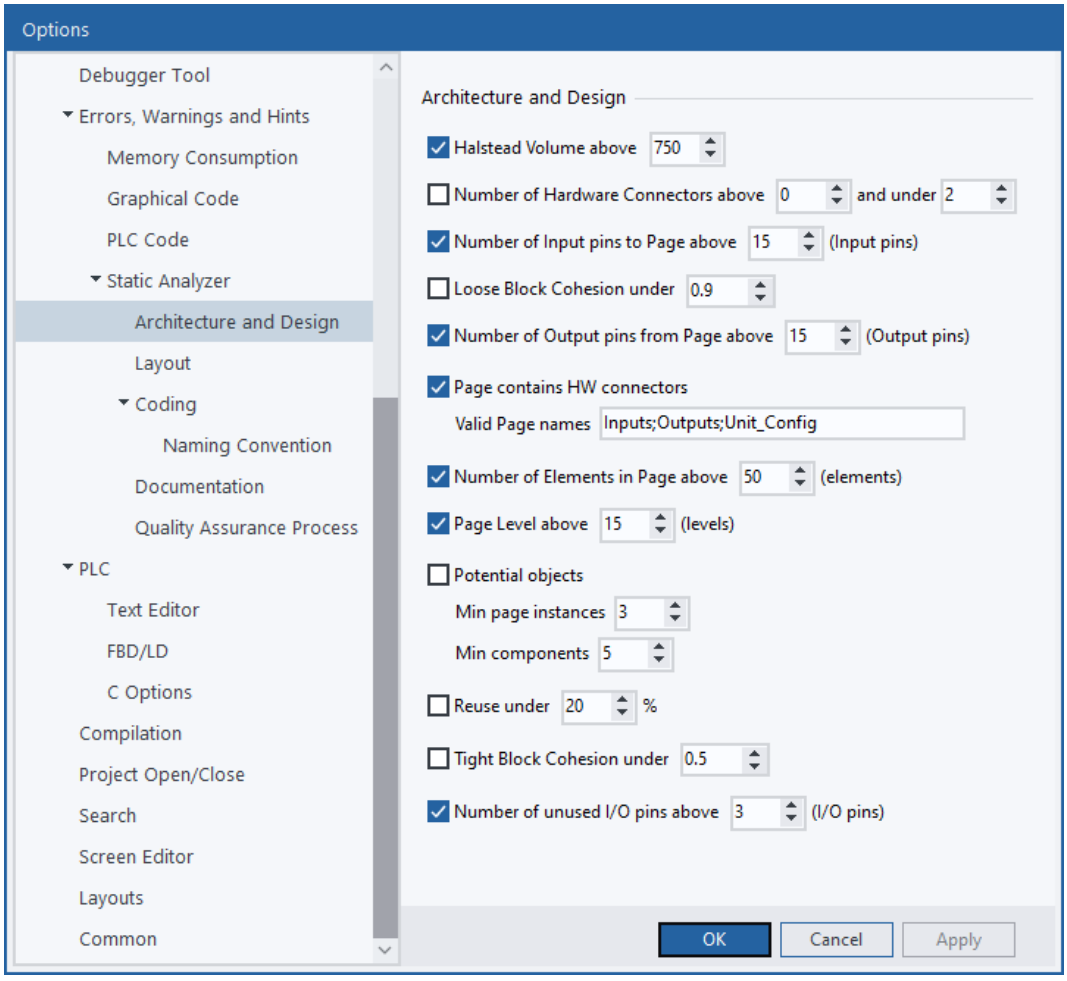


Static Analyzer settings

Item	Description
Export Static Analyzer settings	Exports the current **Static Analyzer** settings to a file in XML format.
Import Static Analyzer settings	Imports **Static Analyzer** settings from a file in XML format.
Enable Full Static Analyzer	**Full Static Analyzer** is executed after each successful check/compile.
Limit warning result	Sets a limit for how many warnings should be reported for each warning type.
Merge warnings from different modules	Warnings of the same type found in different modules will be collected and presented together.
Sort warning result by criticality	Sorts the reported warning result in priority order, with the most critical warnings listed first for each warning type.

User Interface

Architecture and Design

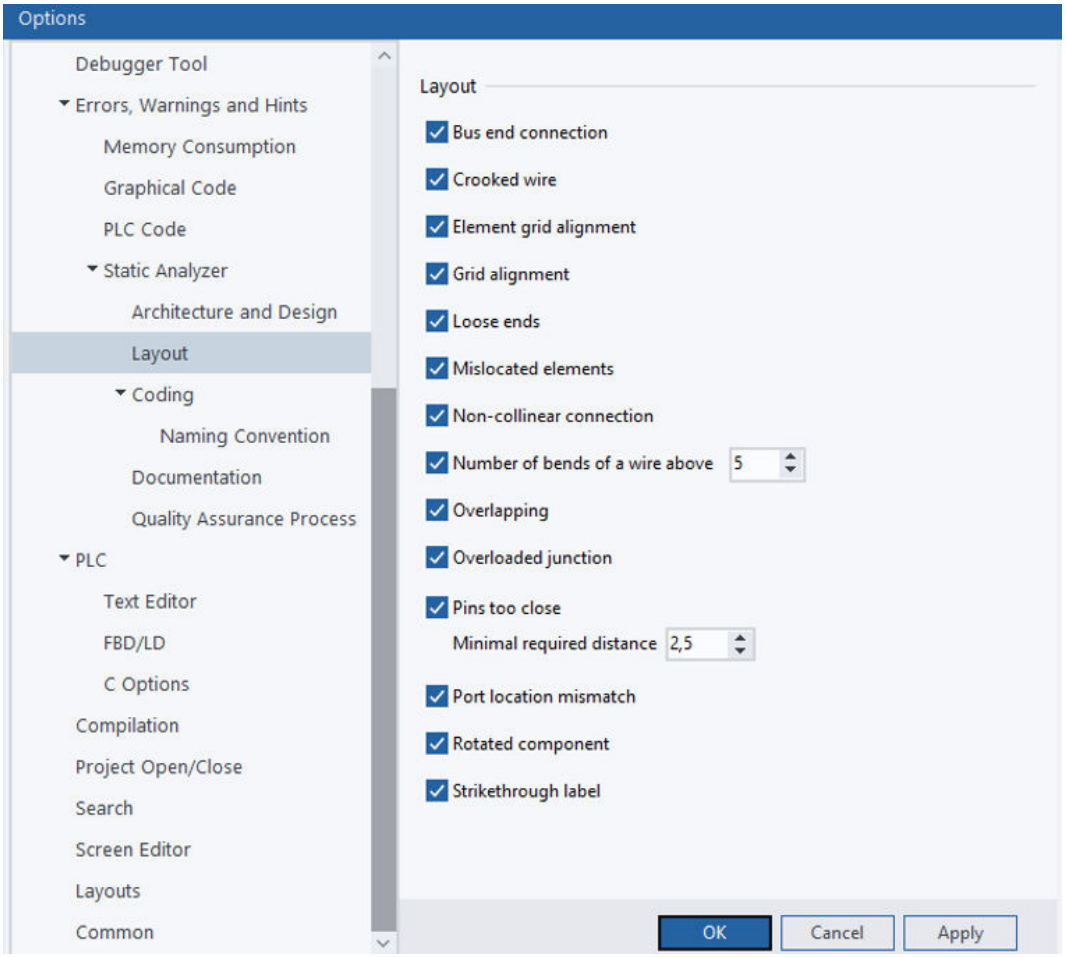


Architecture and Design settings

Item	Description	Warning code
Halstead Volume	Detects complex pages.	662:620
Number of Hardware Connectors	Detects pages containing invalid number of hardware connectors.	662:130
Number of Input pins to Page	Detects pages with too many input pins.	662:140
Loose Block Cohesion	Detects pages with low cohesion.	662:640
Number of Output pins from Page	Detects pages with too many output pins.	662:150
Page contains HW connectors	Detects pages containing any hardware connectors.	662:600
Number of Elements in Page	Detects pages containing too many elements.	662:110
Page Level	Detects pages with high nesting depth/level.	662:120
Potential objects	Detects linked pages convertible to an object.	662:590
Reuse	Detects project/modules with too low reusability.	662:170
Tight Block Cohesion	Detects pages with low cohesion.	662:630
Number of unused I/O pins	Detects pages with too many unused I/O pins.	662:160

User Interface

Layout



Layout settings

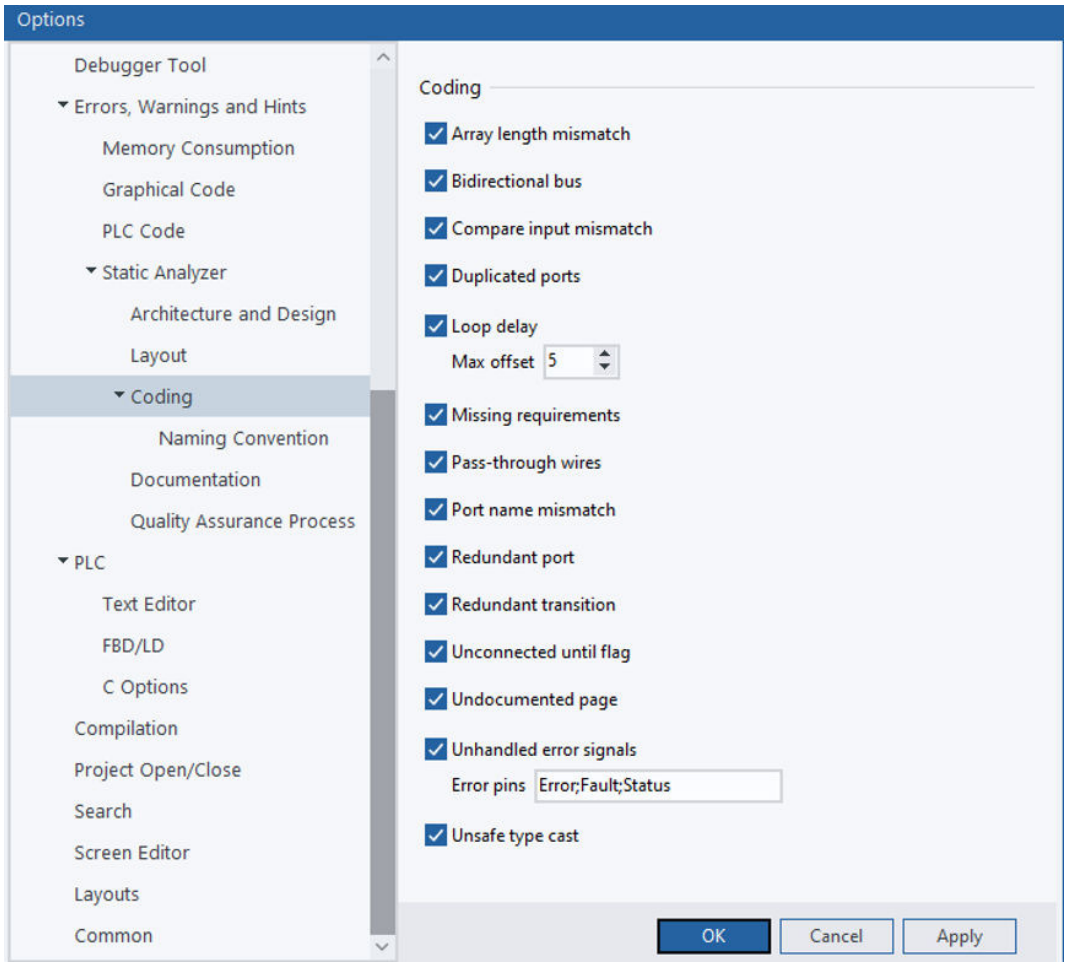
Item	Description	Warning code
Bus end connection	Detects a sub-bus connected to the end of a bus.	662:400
Crooked wire	Detects a route with non-standard line orientations.	662:300
Element grid alignment	Detects elements that are not aligned with the grid lines.	662:750
Grid alignment	Detects page top-view ports (pins) which do not lie on the grid lines.	662:340
Loose ends	Detects routes with dead ends.	662:390
Mislocated elements	Detects elements that are located outside the page drawing area.	662:700
Non-collinear connection	Detects a route that is connected to a pin in a non-collinear way. I.e., the route connected to the pin has a different line orientation than the pin.	662:730
Number of bends of a wire	Detects a route with too many bends.	662:310
Overlapping	Detects overlaps between graphical elements.	662:350
Overloaded junction	Detects a wire node connecting four or more wires.	662:360
Pins too close	Detects if the distance between two connected pins is too small. Set **Minimal required distance** to a lower limit distance (mm).	662:720

User Interface

Layout settings (continued)

Item	Description	Warning code
Port location mismatch	Detects page internal ports located on wrong side within the page	662:370
Rotated component	Detects a component with a non-default rotation.	662:320
Strikethrough label	Detects a route crossing a bus-member label.	662:740

Coding



Coding settings

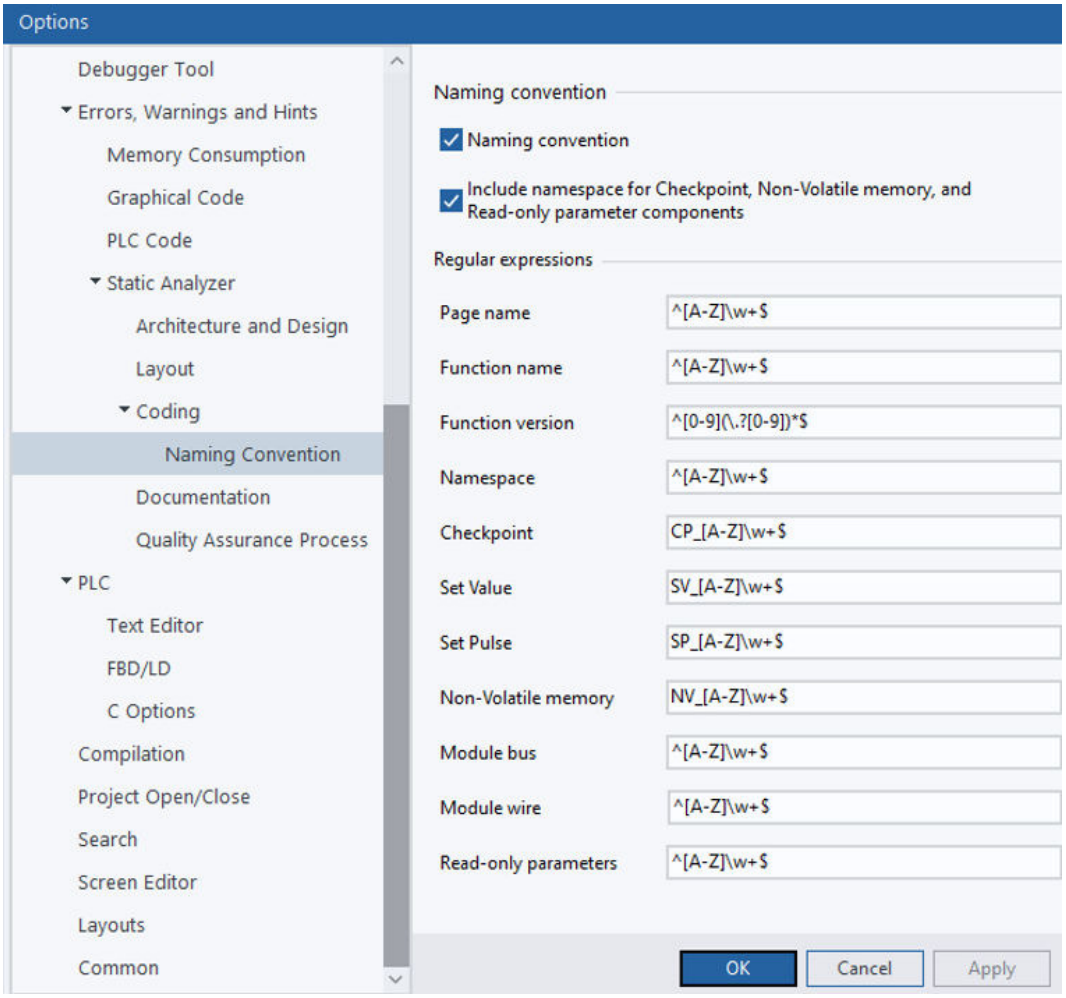
Item	Description	Warning code
Array length mismatch	Detects unreasonable array length change.	662:230
Bidirectional bus	Detects a bus in a page containing both input and output signals.	662:380
Compare input mismatch	Detects unequal input types for compare components.	662:200
Duplicated ports	Detects if a page contains two or more ports with the same name.	662:510
Loop delay	Detects unintentional loop delay caused by incorrect component positioning. Set **Max offset** to a distance (mm) considered as unintentional.	662:240

User Interface

Coding settings (continued)

Item	Description	Warning code
Missing requirements	Detects pages with no requirements (traceability) links.	662:500
Pass-through wires	Detects unused wires passing through a page.	662:710
Port name mismatch	Detects the page top-view ports (pins) where name does not match its label.	662:330
Redundant port	Detects pages that contain two or more ports carrying the same output signal.	662:760
Redundant transition	Detects a duplicated positive transition which has no effect on the program.	662:800
Unconnected until flag	Detects unconnected output signal of an **Until** component.	662:220
Undocumented page	Detects pages with no comments.	662:530
Unhandled error signals	Detects unused output pins possibly carrying error signals. Set **Error pins** to a list of pin names considered to carry error signals.	662:770
Unsafe type cast	Detects unsafe output type for components based on expected output value range.	662:190

Naming Convention

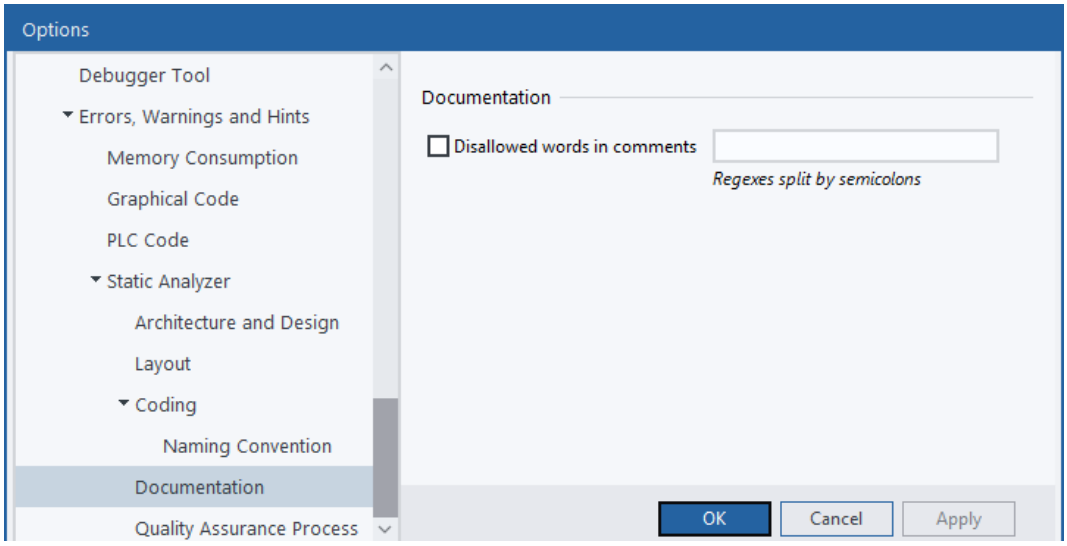


User Interface

Naming convention settings

Item	Description	Warning code
Naming convention	Detects components and their attributes with names not matching naming guidelines (defined as regular expressions).	662:210

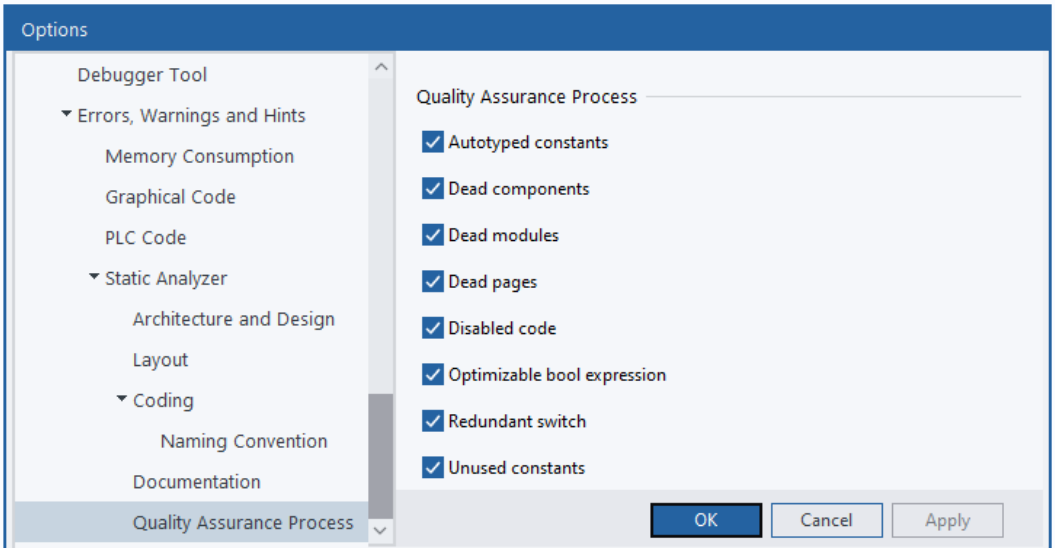
Documentation



Documentation settings

Item	Description	Warning code
Disallowed words in comments	Detects unwanted expressions in the comment texts.	662:540

Quality Assurance Process



User Interface

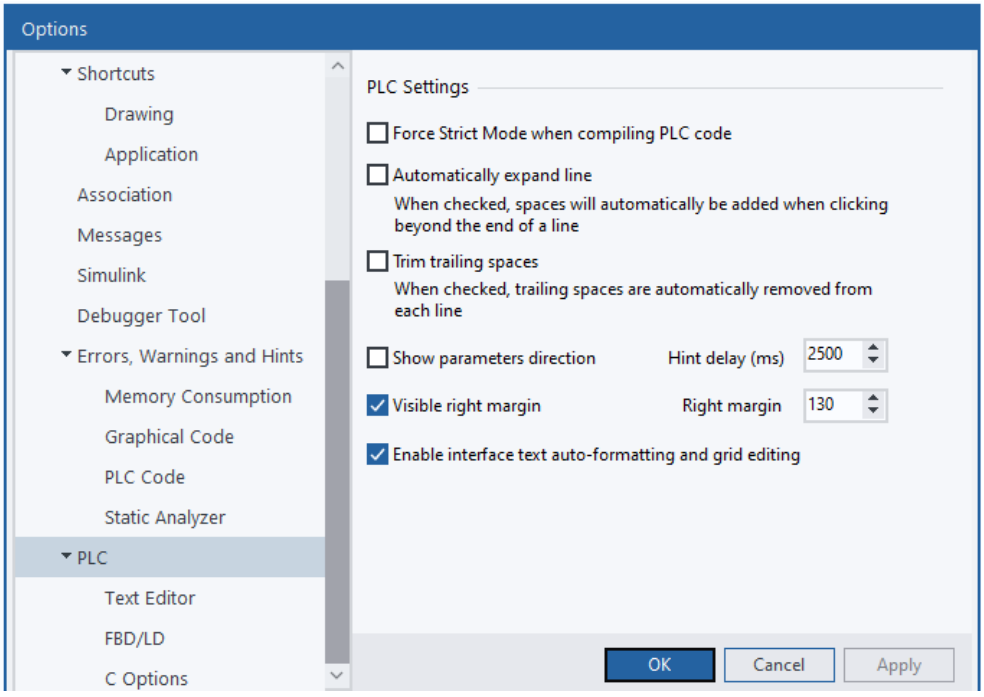
Quality Assurance Process settings

Item	Description	Warning code
Autotyped constants	Detects autotyped constants.	662:250
Dead components	Detects components with no impact to program functionality.	662:100
Dead modules	Detects modules with no impact to program functionality.	662:101
Dead pages	Detects pages with no impact to program functionality.	662:102
Disabled code	Detects pages containing disabled code (Keep In/Keep Out Polygons).	662:520
Optimizable bool expression	Detects redundant logical operation sequences and suggests a more optimal solution.	662:580
Redundant switch	Detects a switch with Boolean input convertible to logical gate.	662:260
Unused components	Detects constants never used.	662:610

User Interface

PLC Settings

Options > **PLC** > **PLC Settings**

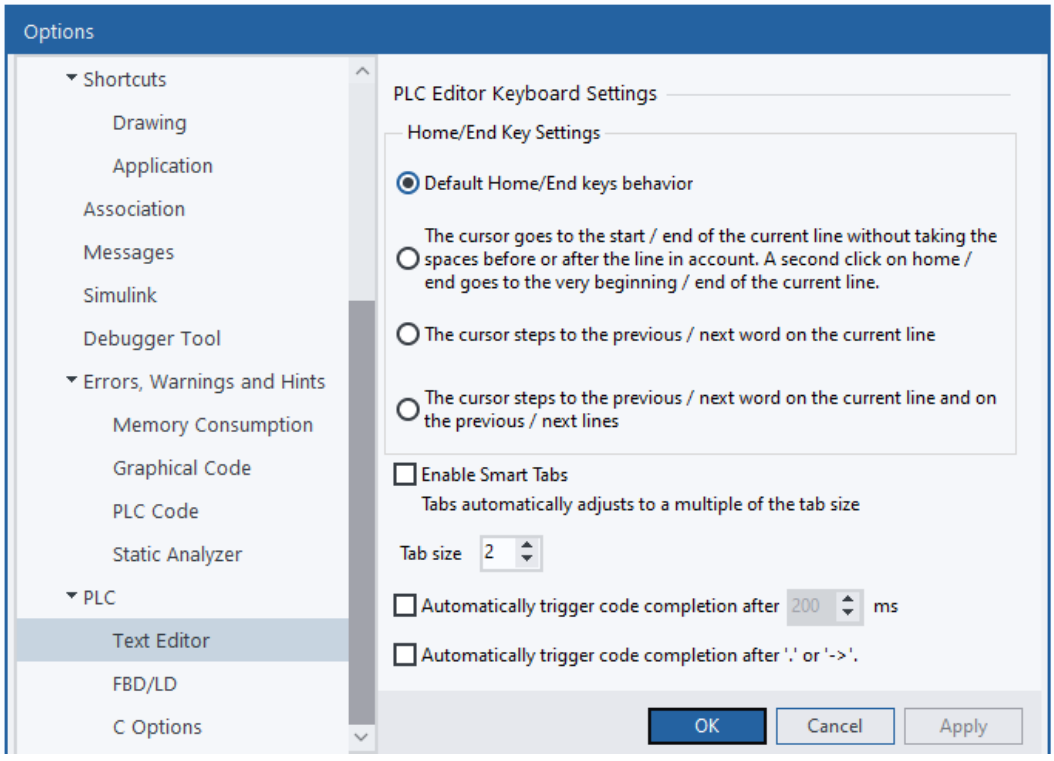


PLC Settings

Item	Description
Force Strict Mode when compiling PLC code	Enforces strict adherence to the IEC 61131-3 standard when compiling PLC code.
Automatically expand line	When checked, spaces will automatically be added when clicking beyond the end of a line.
Trim trailing spaces	When checked, trailing spaces are automatically removed from each line.
Show parameters direction	When using auto completion of a function (**Ctrl + space**), or when editing the parameters of a function, this setting controls if the direction (in/out/in out) is displayed for each parameter of that function.
Hint delay	Controls the duration of the hint displayed when editing parameters of a function.
Visible right margin	Determines whether the right margin is displayed or not.
Right margin	Determines the position of the right margin in number of characters.
Enable interface text auto-formatting and grid editing	When checked, will enable automatic formatting of the text in the interface section of POUs. It also enables the alternative editing of interfaces, using a grid editor.

User Interface

Text Editor Settings



PLC Editor Keyboard Settings Description

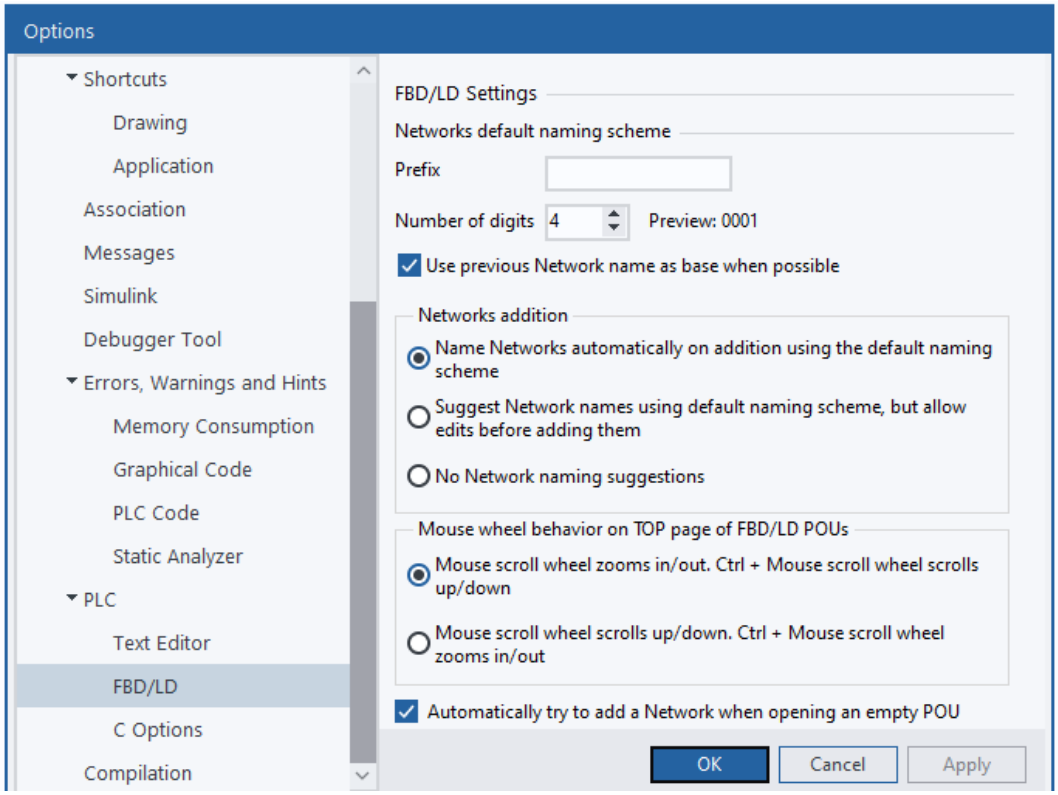
Item	Description
Home/End Key Settings	The behavior of the **Home/End** keys can be configured: • to work as usual • to switch between ignoring • not ignoring white space • to step from word to word • stepping from word to word and then continue from line to line.
Enable Smart Tabs	Enforces multiples of the tabs size
Tab size	The number of space characters corresponding to a tab.
Automatically trigger code completion after *xxx* ms	Automatically triggers code completion at a specific time after you stop typing. Only triggers when there is an incomplete word immediately to the left of the cursor. The time can be set to any value between 200 and 5000 ms.
Automatically trigger code completion after '.' or '->'	Automatically trigger code completion after . or ->. Only triggers when there is a variable of a known structured data type immediately to the left of the . or ->. . Is only active in C and ST code implementation sections and in c/h files. -> Is only active in C code implementation sections and in c/h files.

The **Text Editor** settings apply to code in:

- Structured Text (ST)
- Instruction List (IL)
- C code POUs
- C code Files.

User Interface

FBD/LD settings



FBD/LD settings description

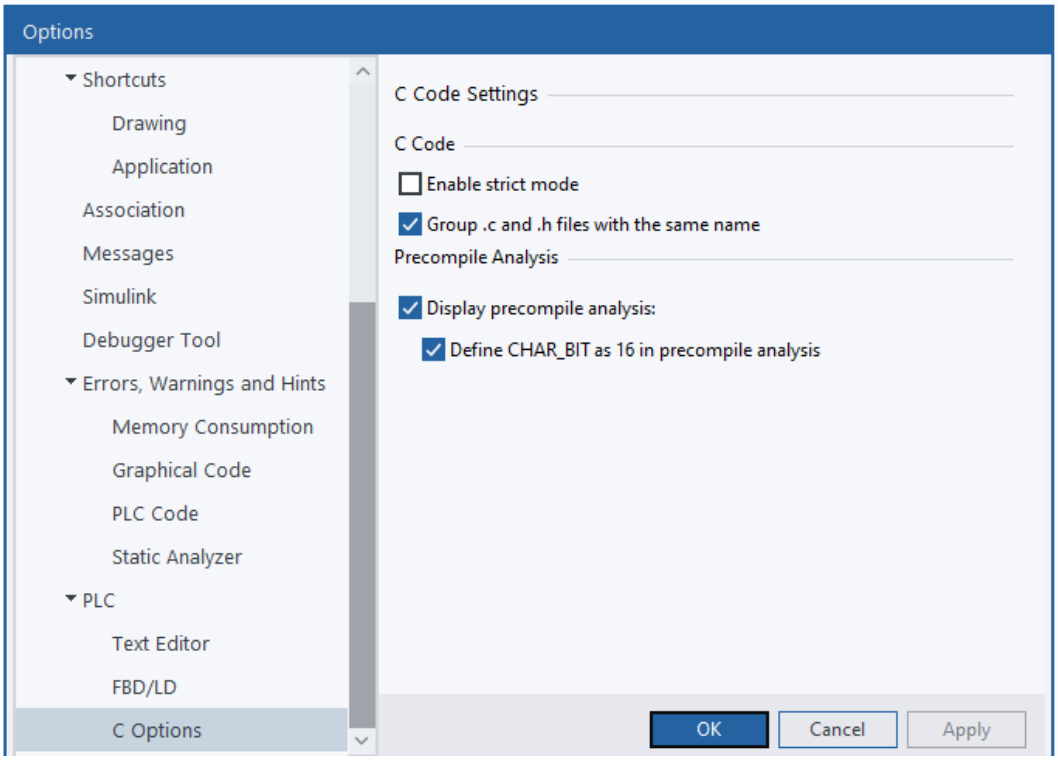
Item	Description
Network default naming scheme	
Prefix	An optional prefix to be used when creating Network page names. *Limitations:* • only ASCII characters • max length = 20 chars • min length = 0 chars. Default text: “” (empty string)
Number of digits	Minimum number of digits to use as ending part of the name in order to make it unique in the POU. (The actual number used will be selected to make the name unique, and is increased by one until the name becomes unique.) *Limitations:* This must be a number between 1 and 5. Default: 4
Preview	This is just a preview of what the first Network page name of a new FBD/LD POU would be named. This text will be updated whenever Prefix or Number of digits change.
Use previous Network name as base when possible	If this box is checked, then the Name of the previous Network page in the FBD/LD POU is parsed and used as prefix/Number of digits when available. Default: Checked
Networks addition	
Name Networks automatically on addition using the default naming scheme	New networks are added immediately. There will be no input dialog to set the name of the new network before it is added. A unique name is created automatically by the tool and used directly. (Default value)

User Interface

FBD/LD settings description (continued)

Item	Description
Suggest Network names using default naming scheme, but allow edits before adding them	A unique name is created automatically by the tool, and set as the preselected value in the input dialog where it is possible to change the name before the page is added.
No Network naming suggestions	The input dialog will not make any naming suggestion when adding network pages.
Mouse wheel behavior on TOP page of FBD/LD POUs	
Mouse scroll wheel zooms in/out. Ctrl + Mouse scroll wheel scrolls up/down	Controls behavior of mouse scroll wheel: • Mouse scroll wheel zooms in/out. • Ctrl + Mouse scroll wheel scrolls up/down. (Default value)
Mouse scroll wheel scrolls up/ down. Ctrl + Mouse scroll wheel zooms in/out	Controls behavior of mouse scroll wheel: • Ctrl + Mouse scroll wheel zooms in/out. • Mouse scroll wheel scrolls up/down.
Automatically try to add a Network when opening an empty POU	When checked, GUIDE will add a new network automatically when opening an empty FBD/LD POU. Default: Checked

C Code Settings



C code settings descriptions

Item	Description
C code	
Enable strict mode	If checked, the C code will be compiled with stricter compiler flags than normal.
Group .c and .h files with the same name	When checked, .c and .h files with the same base name will be grouped in the Project Manager and in the PLC/C Editor. This is just a GUI setting and has no effect on the generated binaries.

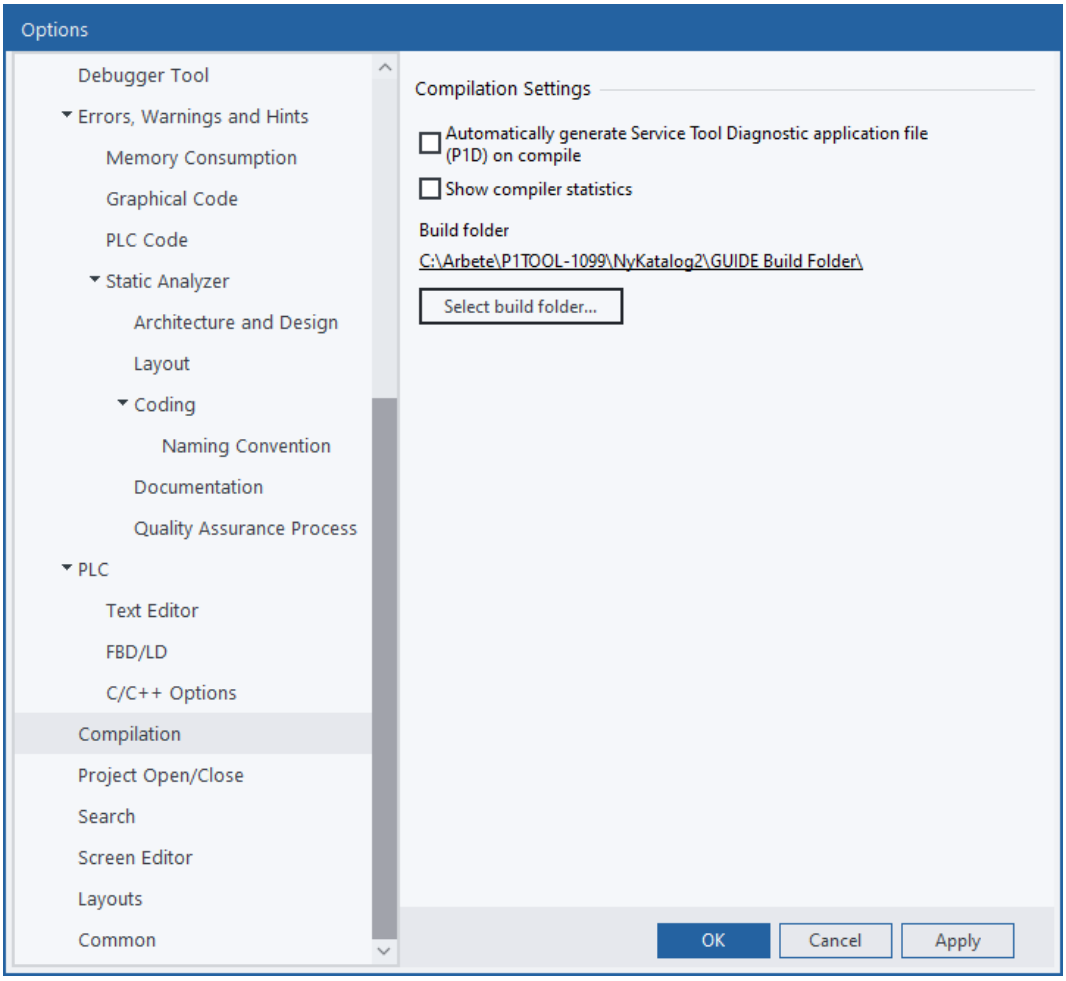
User Interface

C code settings descriptions (continued)

Item	Description
Precompile Analysis	
Display precompile analysis	When checked, a memo with errors and warnings is displayed and updated on save while editing.
Define CHAR_BIT as 16 in precompile analysis	This setting can be used to force a CHAR_BIT value of 16 for the precompile analysis.

User Interface

Compilation Settings

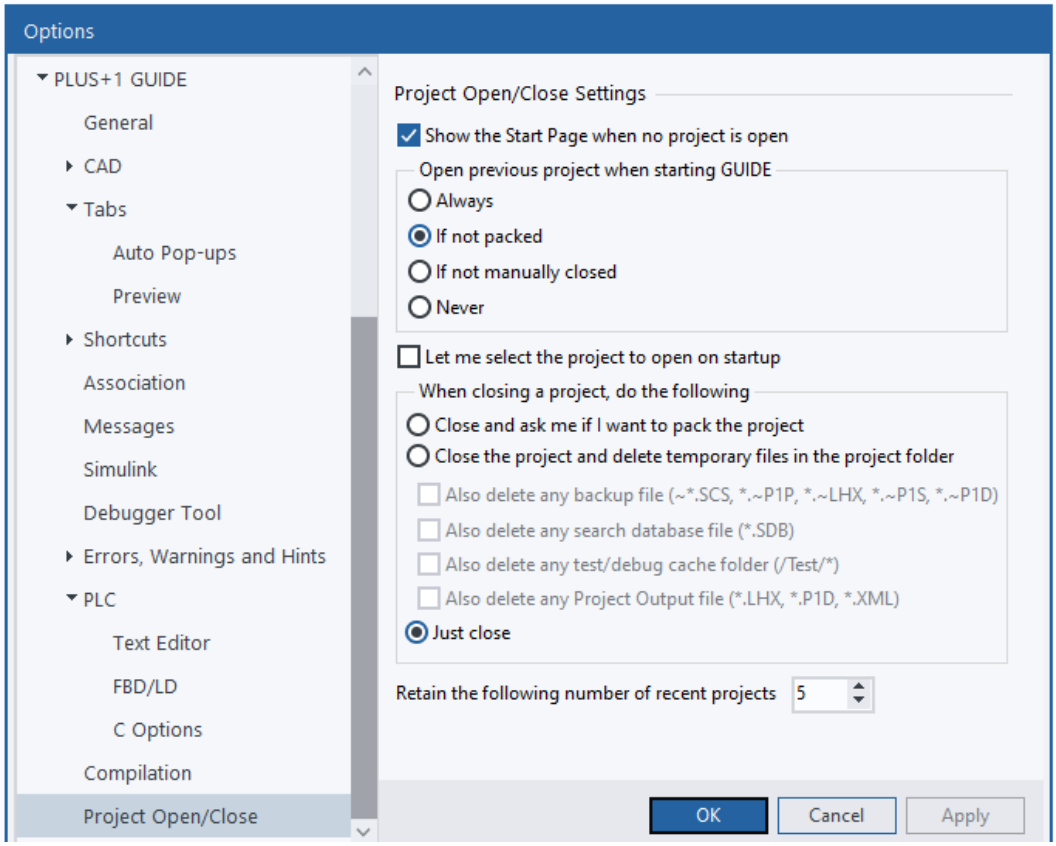


Compilation Settings table

Item	Description
Automatically generate Service Tool Diagnostic application file (P1D) on compile	This feature is intended for use by PLUS+1® GUIDE developers to simplify diagnostics of the application during development. The P1D file is not intended for use by Service Technicians. This feature is currently only for evaluation.
Show compiler statistics	When checked, the **Compile Progress** window displays the **Total pages count** and the **View disabled pages count** of a successfully compiled application. These statistics display in the **User Message** pane of the **Compile Progress** window.
Select build folder	Temporary files generated when PLUS+1® GUIDE applications are built will be located under this folder (requires HWD support). Files added to sub-folders of the build folder may be deleted by GUIDE. Modifying files in sub-folders of the build folder may lead to issues with GUIDE compilation. This includes opening a file in a tool that locks the file. The user must have read and write access for the selected folder, and it must not be located on a network drive. Some GUIDE features require execution rights for the build folder. It is advised to use a folder located on the fastest drive.

User Interface

Project Open/Close Settings



Project Open/Close Settings Description

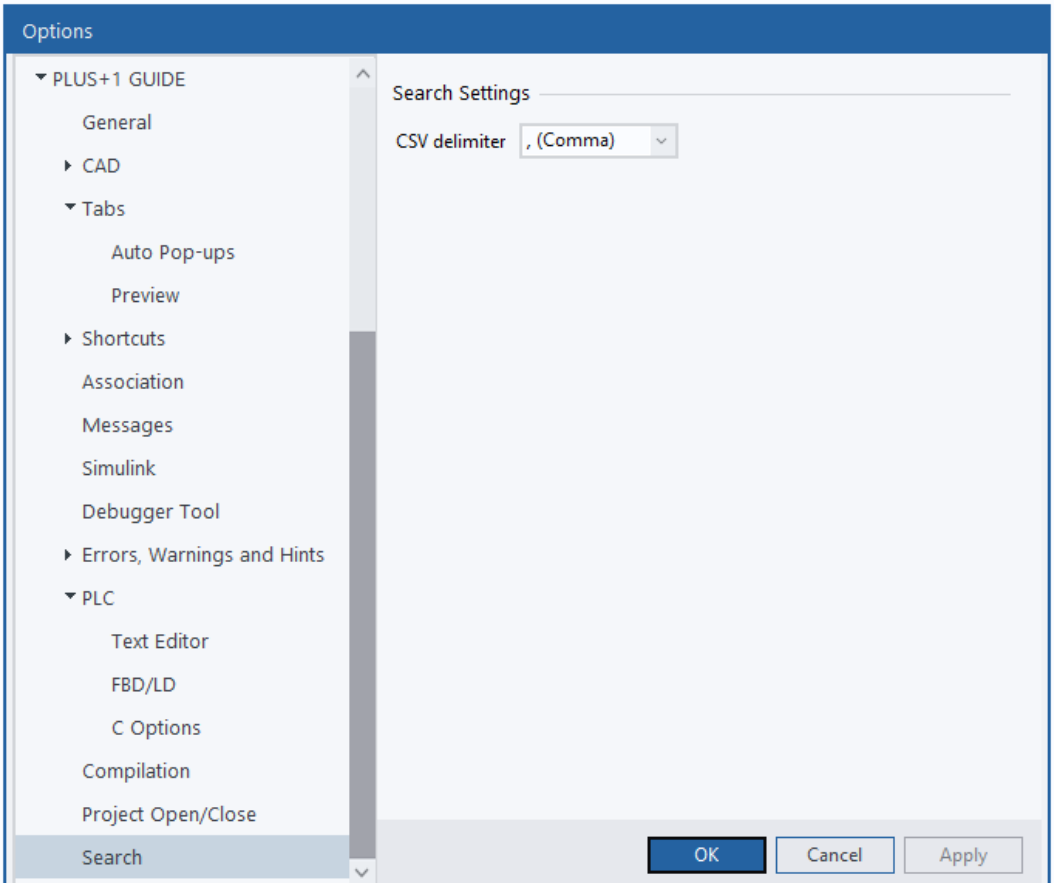
Item	Description
Show the Start Page when no project is open	Determines whether the Start Page will be displayed or not. Default: true.
Open previous project when starting GUIDE	• **Always** — The previous project (if any) will always be opened when GUIDE is started. • **If not packed**— The previous project (if any) will be opened when GUIDE is started, but only if it is in p1x format. • **If not manually closed** — The previous project (if any) will be opened when GUIDE is started, but not if it was previously closed using the "Close project" menu option. • **Never** — The previous project (if any) will never be opened when GUIDE is started.
Let me select the project to open on startup	When checked, GUIDE will display a File Open dialog that lets the user select another project to open, or to cancel the opening of the previous project (if any). If there is a previous project that should be opened, then that file will be pre-selected in the File Open dialog.

User Interface

Project Open/Close Settings Description (continued)

Item	Description
When closing a project, do the following	• **Close and ask me if I want to pack the project** – A dialog will be displayed when a project is closed, giving the option of either packing it to P1P, or not. • **Close the project and delete temporary files in the project folder** – Temporary project files will be deleted. This also gives the option of deleting additional files: — Backup files — Data base files — Test/debug cache files — Output files • **Just close** (Default value) It is recommended to select one of the last two options when working with version control systems like SVN or Git. This setting also affects the behavior for **Project Save As**.
Retain the following number of recent projects	Determines how many Recent Projects that will be available to select from. This includes the currently open project if any. Valid range: 0-20 projects.

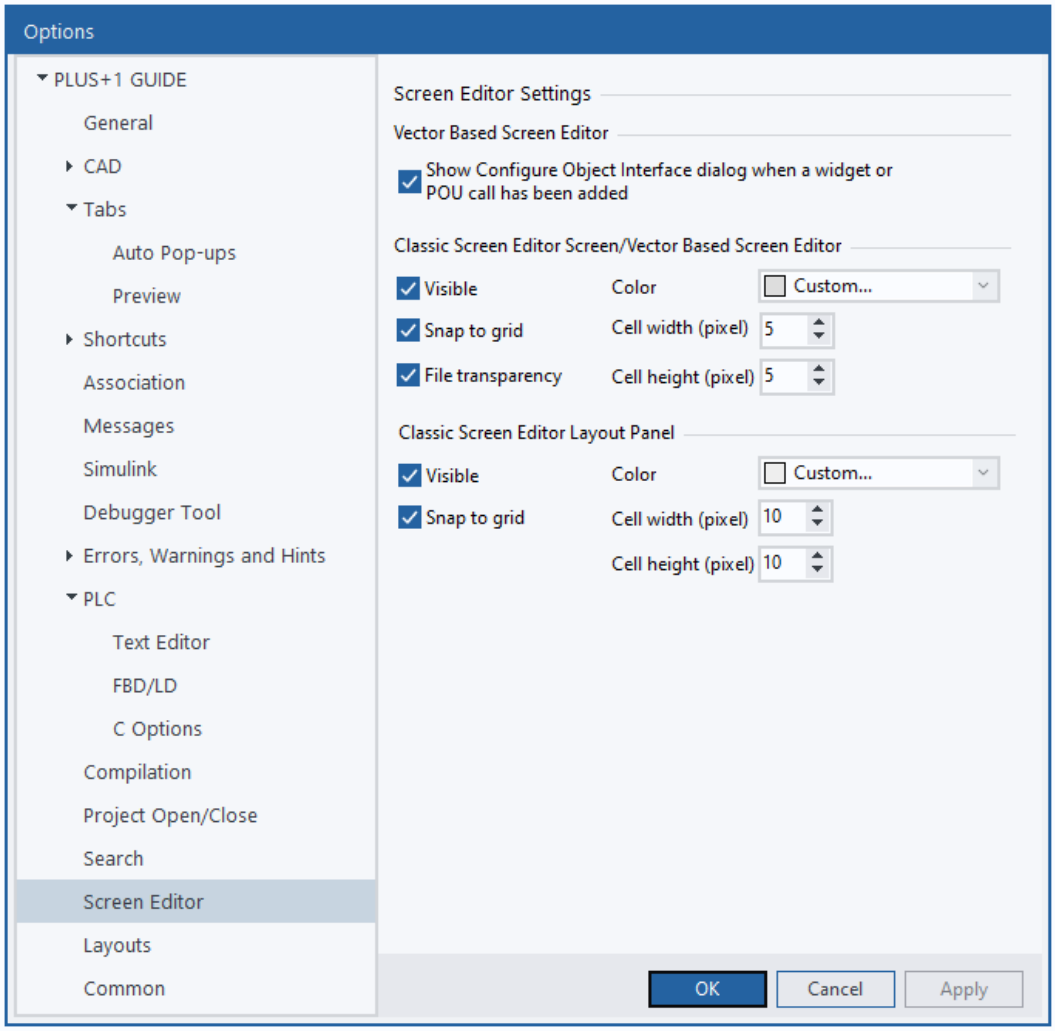
Search Settings



Item	Description
CSV delimiter	The selected **CSV delimiter** will be used when search results are exported to CSV.

User Interface

Screen Editor Settings



Grid Settings define the two grids used in the Screen Editor:

- **Screen Area** grid—this grid appears within each Screen Area. You use this grid in the **Define Screen** page when laying out Screen Library items.
- **Layout** panel grid—this grid fills the entire Layout panel. You use this grid in the **Define Areas** page when laying out Screen Areas.

Options Window—Grid Settings

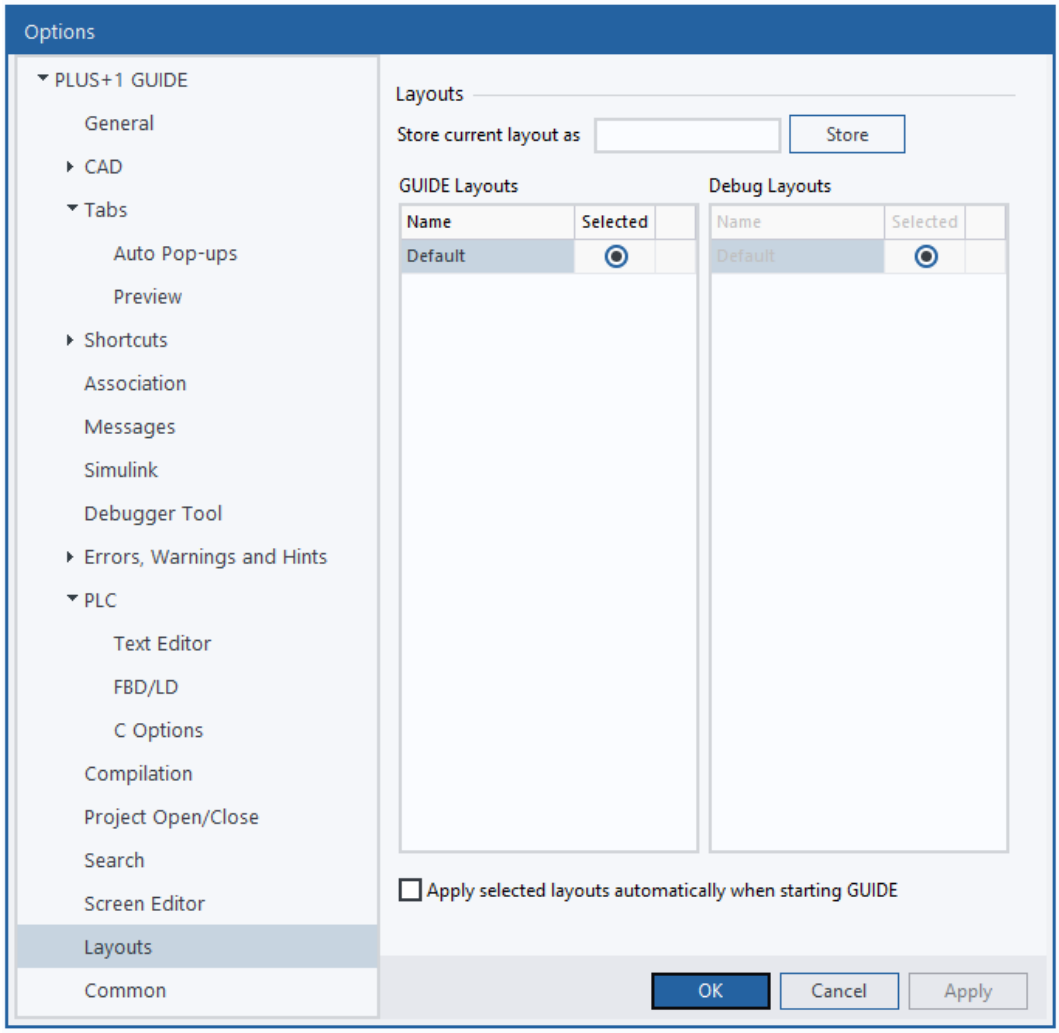
Item	Description
Classic Screen Editor/Vector Based Screen Editor	Sets the characteristics of the grid that appears within each Screen Area/Screen Definition. Use this grid in the **Define Screen page/Vector Based Screen Editor** when laying out items from the **Screen Library**, such as images and text.
Visible	Click to make this grid visible.
Snap to grid	Click to snap the **Reference point** of items to the grid.
File transparency	Click to make all items visible through stacked Screen Areas. Use this option when aligning items placed on different Screen Areas.
Color	Click to select a color for the lines in the grid from a drop-down list.
Cell width (pixel)	Click to set the horizontal distance, in pixels, between gridlines.

User Interface

Options Window—Grid Settings (continued)

Item	Description
Cell height (pixel)	Click to set the vertical distance, in pixels, between gridlines.
Classic Screen Editor Layout Panel	Sets the characteristics of the Layout panel grid that appears within each Screen Area. Use this grid in the **Define Areas** page when laying out Screen Areas.
Visible	Click to make lines in the grid visible.
Snap to grid	Click to snap the upper-left corners of Screen Areas to the grid.
Color	Click to select a color for the lines in the grid from a drop-down list.
Cell width (pixel)	Click to set the horizontal distance, in pixels, between gridlines.
Cell height (pixel)	Click to set the vertical distance, in pixels, between gridlines.

Layouts



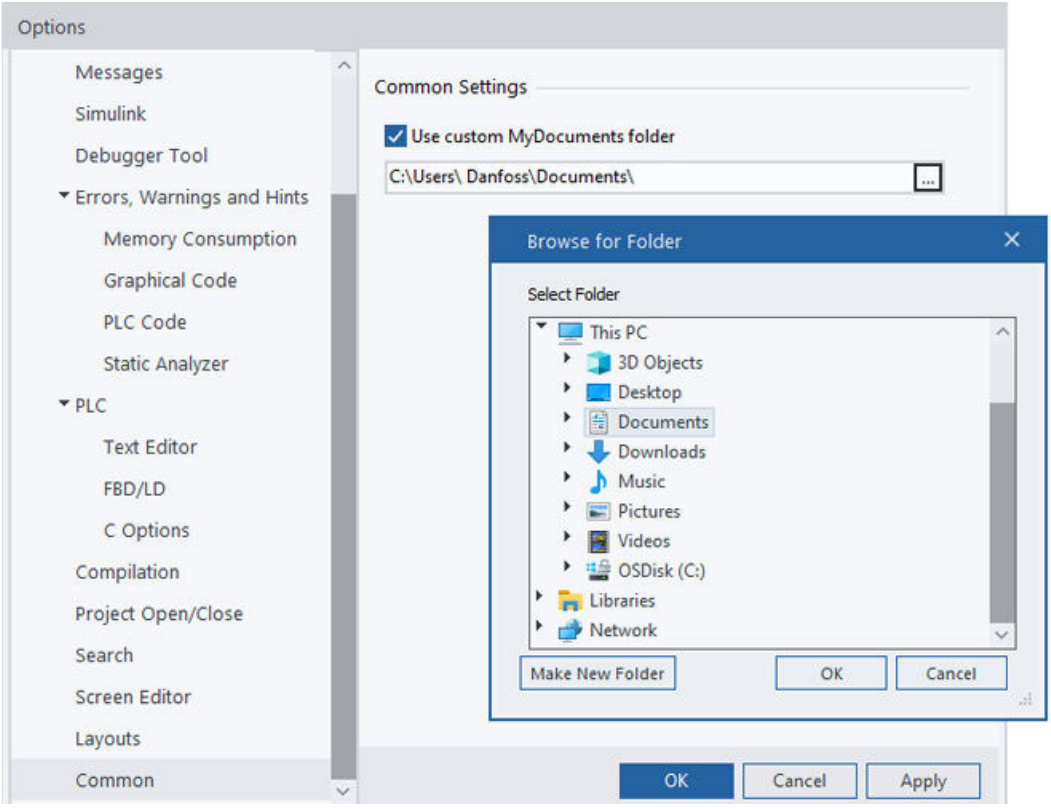
User Interface

Options Window—Layouts

Item	Description
Store Current Layout as	Saves the current positions and sizes of all open non-modal **GUIDE** windows under a specific Layout Name. Stored Layouts are available for use from the toolbar
GUIDE Layouts	A list of available Layouts for use when editing applications in **GUIDE**.
Debug Layouts	A list of available Layouts for use when debugging applications in **GUIDE**.
Apply selected layouts automatically when starting GUIDE	If checked, the selected Layout will be enforced automatically when starting **GUIDE**. If not checked, **GUIDE** will use the same window positions and window sizes as when it was previously closed.

Common Settings

Common Settings



Common Settings

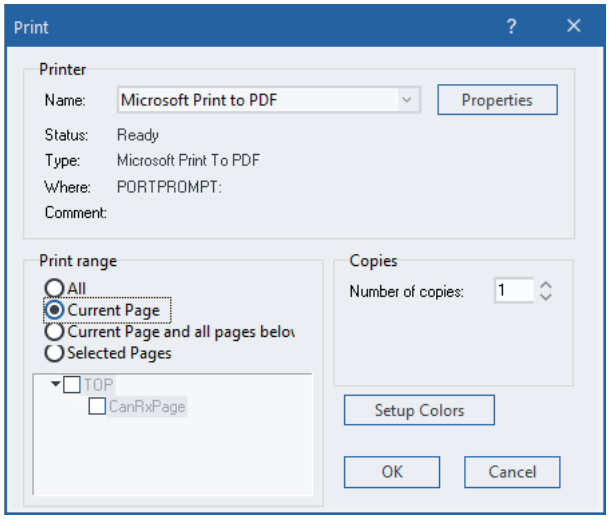
Item	Description
Use custom MyDocuments folder	If checked, the custom **MyDocuments** folder will be used.
Browse for Folder	**Browse for Folder** can be used to select the custom folder path if the check box for **Use custom MyDocuments folder** is checked.

User Interface

Print

File > **Print** command.

Use Print window to print pages from a PLUS+1® GUIDE application.



Print window description

Item	Description
Name	Selects a printer.
Properties	Displays a **Printer Properties** window. Use this window to enable features specific to the selected printer. Each printer model has a different **Printer Properties** window.
Copies	Sets the number of copies made of each page.
Print range	• **All**—prints all pages in the application. • **Current page**—prints the page now displayed in the PLUS+1 GUIDE window's Drawing Area. • **Current page and all pages below**—prints the current page and all pages within the current page. • **Selected pages**—prints the pages that have been checked in the Print Selection Tree.
Print Selection Tree	Displays a tree view of all the pages in the application. To choose pages for printing, select the **Pages** option and check the pages that you want printed.
Setup Colors	Displays the **Assign Pens** window. Use this window when printing in color to remap the default colors assigned to items in the Drawing Area.

User Interface

Project View

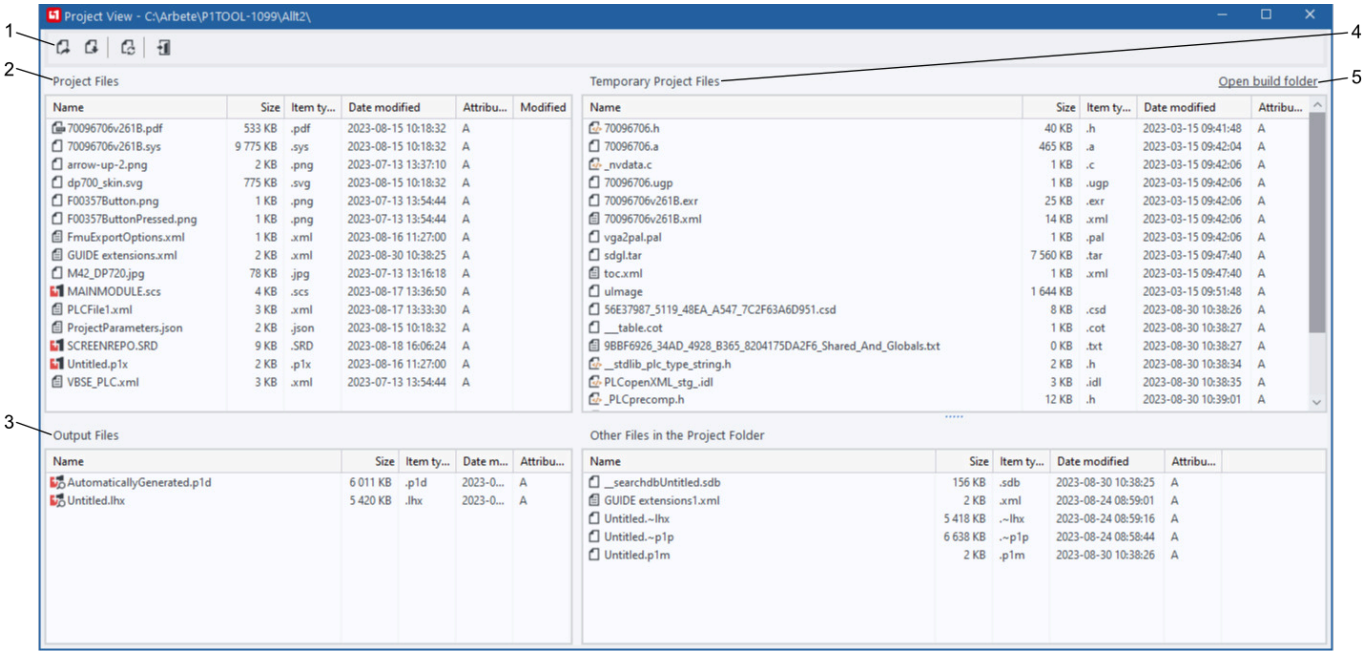
File > **Project View** window

This is a PLUS+1® GUIDE upgrade feature. A Quality Assurance License (Reference PLUS+1® GUIDE add on license Quality Assurance Data Sheet, **AI170686484256**) enables this feature. For more information, see PLUS+1® GUIDE Licensing, **AQ419969404812**.

Use this window in combination with a third-party version-control program such as Apache™ Subversion® to:

- Identify the core files that you need to open, continue, and compile a project.
- Export these core files for safekeeping to your version-control repository.

Your version control program implements the export functions that are available through this window and probably will change the appearance of the file icons in this window.



Project View Window Description

Item			Description
1.		Export Project to 7z Zip	Opens a **Save As** window, export the files that are listed in the **Project Files** tab to either a ZIP or 7z archive format file.
		Generate a Report	Opens a **Save As** window, save a TXT format file that reports project-related items such as the: • Project path. • Names of project files. • Names of temporary project files. • Names of non-project files. • Names of referenced files.
		Refresh	Refresh the lists of files shown in the **Project View** window.
		Close	Closes the **Project View** window.
2.	**Project Files** tab		Lists the core files that you need to open, continue, and compile a project. These are the files that you should export for safekeeping to your version-control repository.

User Interface

Project View Window Description (continued)

Item		Description
3.	**Output Files**	Lists files—such as LHX files, that can be recreated from the core files that are listed in the **Project Folder** tab. **Project Folder** > **Other Files** This tab may also contain files that are unrelated to your project.
4.	**Temporary Project Files**	Lists the temporary files that the PLUS+1® GUIDE software needs to compile an LHX file. The PLUS+1® GUIDE software builds these temporary files out of the core files that are listed in the **Project Folder** tab.
5	**Build folder link**	Open the build folder in **File Explorer**. Only available when the application uses a separate build folder.

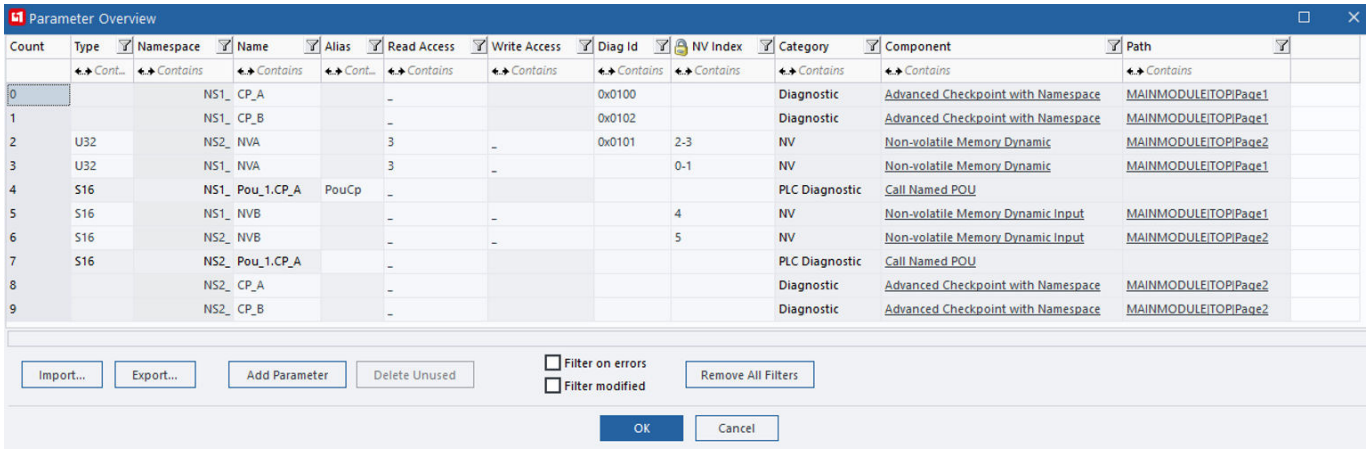
Parameter Overview

The Parameter Overview dialog provides a list of all Diagnostic and Non-volatile parameters in the current application.

It is possible to change editable properties for these parameters.

It is also possible to add new parameters that can be used later when querying Diagnostic and Non-volatile Components.

All columns can be sorted and filtered, and it is possible to copy text from any cell.



Count	Type	Namespace	Name	Alias	Read Access	Write Access	Diag Id	NV Index	Category	Component	Path
0		NS1_	CP_A		_		0x0100		Diagnostic	Advanced Checkpoint with Namespace	MAINMODULE\TOP\Page1
1		NS1_	CP_B		_		0x0102		Diagnostic	Advanced Checkpoint with Namespace	MAINMODULE\TOP\Page1
2	U32	NS2_	NVA		3	_	0x0101	2-3	NV	Non-volatile Memory Dynamic	MAINMODULE\TOP\Page2
3	U32	NS1_	NVA		3	_		0-1	NV	Non-volatile Memory Dynamic	MAINMODULE\TOP\Page1
4	S16	NS1_	Pou_1.CP_A	PouCp	_				PLC Diagnostic	Call Named POU	
5	S16	NS1_	NVB		_	_		4	NV	Non-volatile Memory Dynamic Input	MAINMODULE\TOP\Page1
6	S16	NS2_	NVB		_	_		5	NV	Non-volatile Memory Dynamic Input	MAINMODULE\TOP\Page2
7	S16	NS2_	Pou_1.CP_A		_				PLC Diagnostic	Call Named POU	
8		NS2_	CP_A		_				Diagnostic	Advanced Checkpoint with Namespace	MAINMODULE\TOP\Page2
9		NS2_	CP_B		_				Diagnostic	Advanced Checkpoint with Namespace	MAINMODULE\TOP\Page2

The Parameter Overview dialog is opened by double clicking on "Parameter Overview" in the Project Manager tree.

Parameter Overview table

Column	Editable	Description
Count	No	This is just a count to indicate the number of parameters.
Type	Yes	This is the parameter type in cases where the parameter is connected to the Component output. Empty otherwise.
Namespace	No	If there is a Namespace, and the Component used is Namespace-enabled, then this shows the Namespace part of the parameter name. Empty otherwise.
Name	Yes	This is the parameter name, excluding the Namespace part where applicable. Note that Parameter names must start with an upper-case character.
Alias	Yes	This is the alias which replaces the parameter name in Service Tool. Only editable for parameters originating from CCPs and POUs. If empty, the original parameter name is used.
Read Access	Yes	This is the read access level when logging this parameter with the Service Tool. Valid values are 0-9 and underscore '_'.
Write Access	Yes	This is the write access level when changing this parameter with the Service Tool. Valid values are 0-9 and underscore '_'. Only applicable for writeable parameters.

User Interface

Parameter Overview table (continued)

Column	Editable	Description
Diag ID	Yes	Allows setting Diagnostic ID according to the UDS standard in cases where this is required. Only values reserved in the UDS Standard as Vehicle Manufacturer Specific are possible to use. Defining Diagnostic IDs will make diagnostics slightly less efficient when logging many signals at the same time in the PLUS+1® Service Tool. Requires HWD support.
NV Index	Protected	Use the lock icon in the column header to toggle between read-only mode and editable mode for this column. This allows to change the NV index or indexes where an NV parameter is stored. At first use, the indexes will automatically be assigned to the ones that would have been used on compile in older GUIDE versions, but then the index positions will only change as a reaction to some user interaction, such as: • User changes the value in this column. • User changes the type of an NV parameter to a bigger type that requires more NV space. In all other cases, the NV parameters will stay put in the same place from one compilation to the next. Note1: Renaming an NV parameter is functionally equivalent to deleting it and creating a new one. The NV index is not retained. Note2: Each NV index represents 16 bits of NV memory. This means that: • U32 and S32 parameters will need 2 indexes each. • In cases where NV memory is scarce, it can be economical to pack U8, S8 and BOOL parameters together before storing them to NV.
Category	No	NV, Diagnostic, CCP Diagnostic, CCP NV, PLC Diagnostic, PLC NV, Global PLC Diagnostic or Global PLC NV.
Component	No	The name of the used component. This is a clickable link that shows the position of the parameter in the application.
Path	No	Module/Page path to the used component. This is a clickable link that shows the position of the parameter in the Module Viewer.
<Unnamed last column>	No	Provides a delete button for New Parameters that are not yet in use.

Parameter Overview buttons

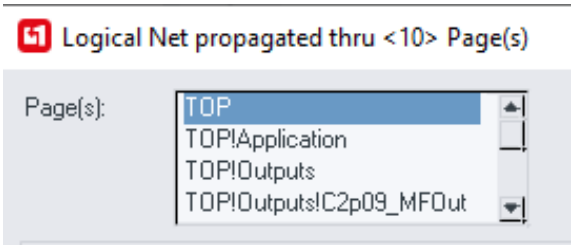
Button	Description
Import…	Import a previously exported Parameter Overview CSV file.
Export…	Export the contents of the Parameter Overview to a CSV file.
Add Parameter	It is possible to add New Parameters, with optional type and access levels. These New Parameters can later be used when Querying NV and Diagnostic Components to quickly fill in the corresponding attributes.
Delete Unused	Deletes all New Parameters that have not yet been used in the application.
Filter on errors	If checked, only lines containing validation errors will be displayed.
Filter modified	If checked, only lines containing changes to be applied will be displayed.
Remove All Filters	Clears all filters in the Parameter Overview table.
OK	Close the dialog and apply any changes that may have been made since it was opened. (Changed lines will be shown in a **bold** font.) This button will be disabled if there are any parameter validation errors.
Cancel	Close the dialog and discard any changes that may have been made since it was opened.

Logical Net

View > **View Logical Net**

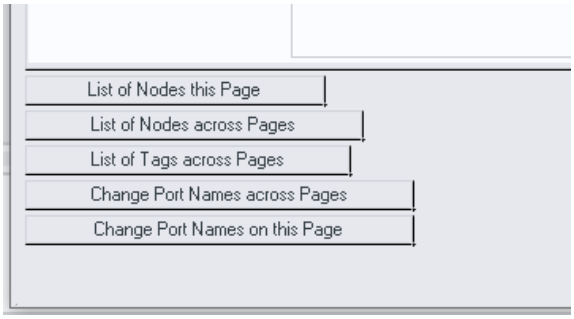
Trace the connections of a selected route on a single page or throughout multiple pages.

User Interface



Logical Net window description

Item	Description
Page(s)	Lists the pages where connections are made to the selected route. Click a page name in **Page(s)** list to display the page in Page Preview tab.
Page Preview tab	Previews the page selected in the **Page(s)** list, highlighting the selected route and its connections. Click in this pane to close the **View Logical Net** window and display the previewed page in the Drawing Area.



Item	Description
List of Nodes this Page	Not implemented in this release.
List of Nodes across Pages	Not implemented in this release.
List of Tags across Pages	Not implemented in this release.
Change Port Names across Pages	Click to display the **Define New Name for <n> Ports** window (<n> is the total number of ports to be renamed). The name that you enter in this window replaces all the port names that belong to the selected route. The **Define New Name for <n> Ports** window renames the ports on every page in which the selected route appears. This window does not rename the ports that bring signals into and out of pages.
Change Port Names on this Page	Click to display the **Define new Port Name** window. The name that you enter in this window replaces just the port names in the route that is highlighted in the Page Preview tab. The Define new Port Name window only renames ports on the page displayed in the Page Preview tab. This window does not rename the ports that bring signals into and out of the page.

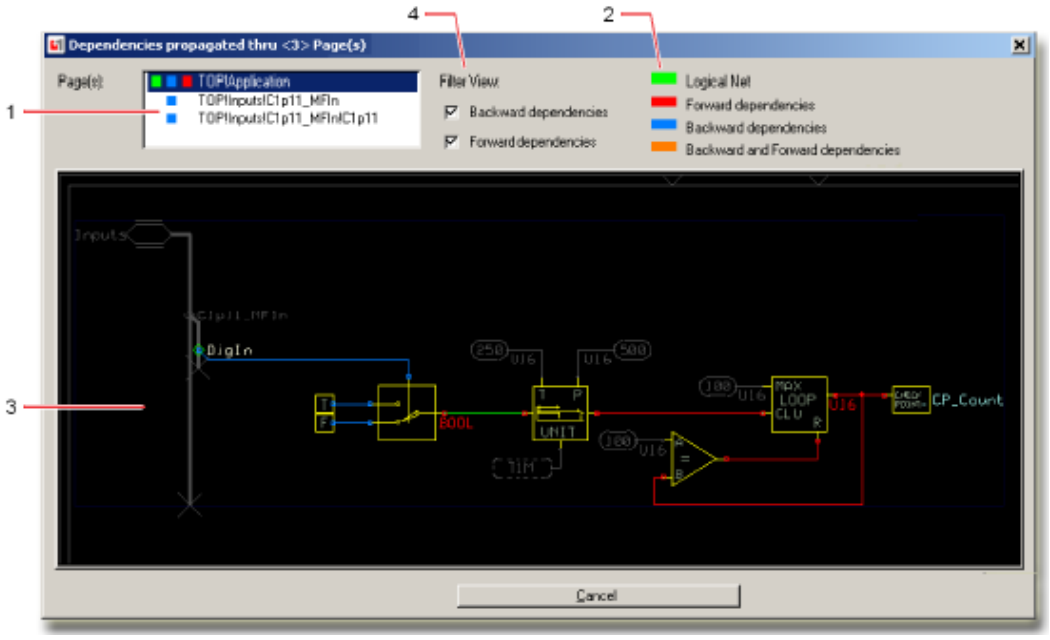
Dependencies

View > **View Dependencies**

Trace the:

- Logical net connections of a selected route.
- Forward dependency connections of a selected route.
- Backward dependency connections of a selected route. You can trace the selected route on a single page or through multiple pages.

User Interface



This is a PLUS+1® GUIDE upgrade feature. A Quality Assurance License (Reference PLUS+1® GUIDE add on license Quality Assurance Data Sheet, **AI170686484256**). For more information, see PLUS+1® GUIDE Licensing, **AQ419969404812**.

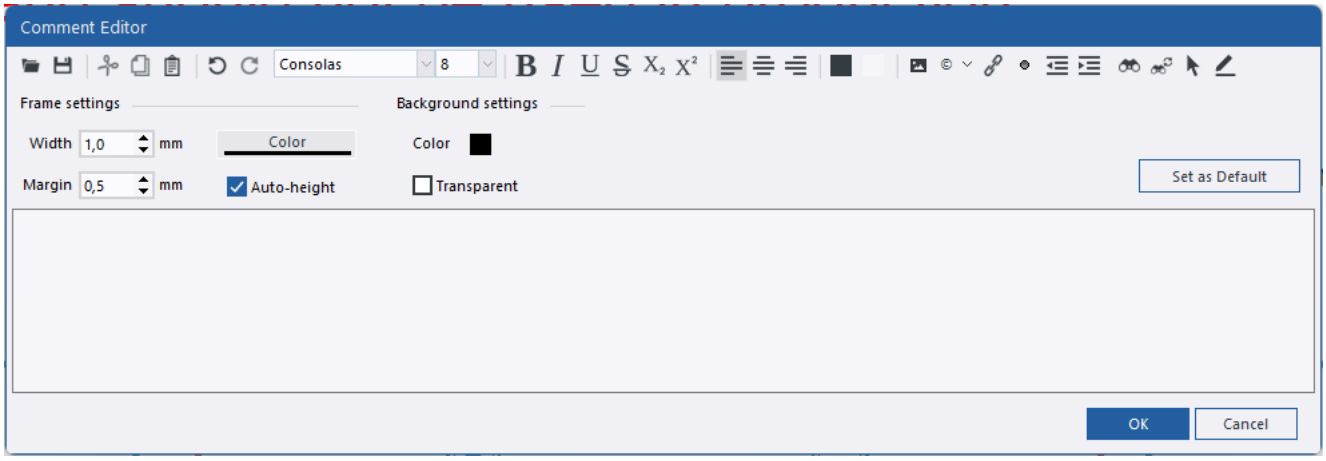
Dependencies window description

Item		Description
1.	**Page(s)**	Lists the pages that have routes with: • Logical net connections to the selected route. • Forward dependency connections to the selected route. • Backward dependency connections to the selected route. Click a page name in the **Page(s)** list to display the page in **Page Preview** panel.
2.	Connection type	Colors indicate the types of connections made on each page: – Routes on the page make logical net connections to the selected route. – Routes on the page make forward dependency connections to the selected route. – Routes on the page make backward dependency connections to the selected route. – Routes on the page combine a forward and a backward dependency connection to the selected route.
3.	**Page Preview**	Previews the page selected in the Page(s) list, highlighting the selected route and its connections. Click in this pane to close the Dependencies window and display the previewed page in the Drawing Area.
4.	**Filter View**	• **Forward dependencies check box** — check to highlight routes in the Preview pane that have forward dependency connections. Clear the Forward dependencies check box to: — Dim routes in the Preview pane that have forward dependency connections. — Hide pages in the Page(s) list that only have forward dependency connections. • **Backward dependencies check box** — check to highlight routes in the Preview pane that have backward dependency connections. Clear the Backward dependencies check box to: — Dim routes in the Preview pane that have backward dependency connections. — Hide pages in the Page(s) list that only have backward dependency connections.

User Interface

Comment Editor

The Comment Editor allows for adding multiple lines of Unicode comments with image support into the drawing area.



Use this dialog to insert structured comments that may contain structured text, images and hyperlinks.

Detailed description

Button		Description
	Open	**Open** a document
	Save	Opens a **Save-as** dialog for rtf-files. (rtf, pdf, html and txt)
	Cut	**Cut** selected text and/or images
	Copy	**Copy** selected text and/or to clipboard
	Paste	Pastes previously copied text and/or images
	Undo/Redo	**Undo** — reverses programming actions. **Redo** — reverses **Undo** actions.
Consolas	Font selection	Selects active font
8	Size selection	Select size of font
B	Bold	Selected text becomes **Bold**
I	Italic	Selected text becomes ***Italic***
U	Underline	Selected text becomes **Underline**
S	Strike through	Selected text becomes **Strike through**
X_2	Subscript	Selected text becomes **Sub**$_{\textbf{script}}$
X^2	Superscript	Selected text becomes **Super**$^{\textbf{script}}$
	Alignment of text	Select **alignment** of test: Left, Mid or Right.

User Interface

Detailed description (continued)

Button		Description
	Text color	Select **color** of selected text.
	Background color	Select **background color** in editor.
	Insert picture	Opens a dialog to select a **picture**.
	Select character	Select a special character.
	Insert Hyperlink	Selected text is designated as a **hyperlink**.
	Insert Bullet	Select **Bullet** list.
	Start List	Starts a **list**, numbered or bullet.
	Decrease/Increase intent	Decreases and increases **indentation**.
	Find	Opens a **Find** dialog.
	Search and Replace	Opens a **Search and Replace** dialog.
	Select All	**Selects All** in edit field.
	Highlight	Opens a **Highlight** text dialog

User Interface

Search/Replace

Edit > **Search…** command.

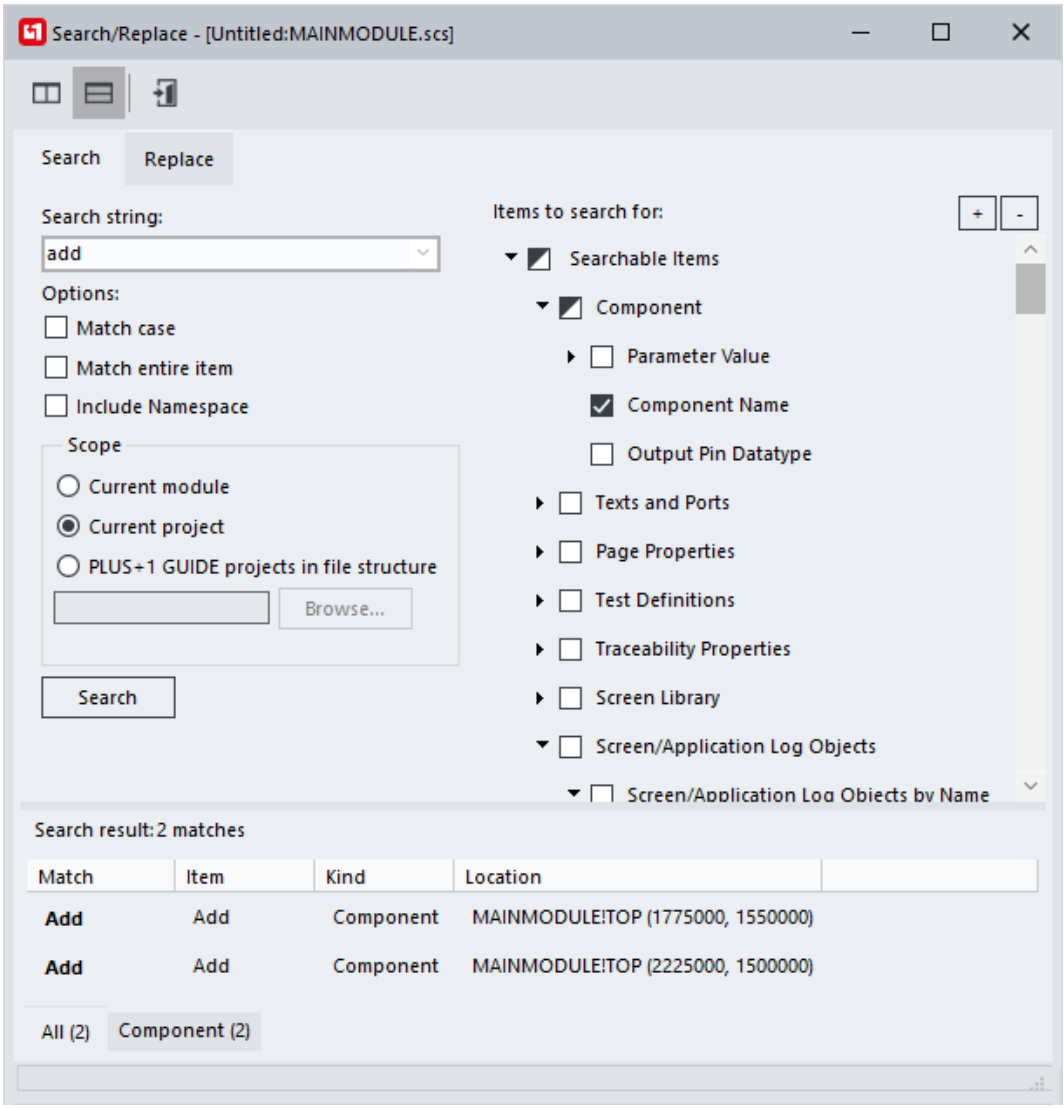
The **Search/Replace** window has two tabs; one for search functionality and one for replace functionality.

Search

Use **Search** functionality to search for items in the currently opened project or inside PLUS+1® GUIDE projects in a file structure.

Search results will be shown in the search result view where each search result has a hyperlink. Clicking a search result will follow the hyperlink, opening PLUS+1® GUIDE windows as necessary. If a hyperlink targets a search hit inside another PLUS+1® GUIDE project than the currently opened one, that PLUS+1® GUIDE project will be opened.

Search/Replace dialog - Search tab



User Interface

Search/Replace dialog - Search tab

Name	Description
Align Horizontally **Align Vertically**	Click on the buttons **Align Horizontally** and **Align Vertically** to choose alignment of the two main components.
Close	Click on this button to close the **Search/Replace** dialog.
Search string	Enter text into **Search string**. If no text is specified, all searchable items will be found.
Options	Select **Options** choices to define how **Search string** will be matched against searchable items. See *Options* on page 82 for a detailed description.
Scope	Select **Scope** choices to define the scope of the search. See *Scope* on page 97 for a detailed description.
Searchable items	Use the **Searchable Items** tree under the **Items to search for** tab to select which items to include in a search. See *Searchable Items Tree* on page 83 for a detailed description.
Search	Click on the **Search** button to search for items using the specified **Search string**, **Options**, **Scope**, and **Items to search for**. After search has finished, the **Search result** view will be updated. A **Progress** dialog appears if the search will be time-consuming.
Search result	View all the search results. See *Search Result View* on page 90 for a detailed description.
Title bar	The text within brackets shows the scope of the search on the format [Project name:unit name]. Project name is the name of the project being searched if **Scope** is set to “Current module” or “Current project”; unit name indicates the scope of the search when **Scope** is set to “Current module.”

Options

Options

Item	Behavior when checked
Match case	Search will be case sensitive. For example, if **Match case** is checked, a search for **Search string** `Add` will match **Add** components, while a search for component name `add` will not match any **Add** components. If this option is unchecked, a search for `add` will find **Add** components.
Match entire item	The entire value of the searchable item will be matched against **Search string**. For example, if **Match entire item** is checked, a search for **Search string** `Add` will only find **Add** components. If this option is unchecked, both **Add** and **Add Capped** components will be found.
Include Namespace	When searching for components with Namespace, the value including Namespace will be matched against Search string. This option is ignored in all other cases. For example, if a page with Namespace Engine_Hours contains an Advanced Checkpoint with Namespace with the name Minutes, the full name of the match including Namespace will be Engine_Hours_Minutes. A search for engine with **Include Namespace** checked and checkboxes **Match case** and **Match entire item** unchecked will find the checkpoint. If **Include Namespace** is unchecked, the match will not be found.

Namespace Components

Matching of the components in the list below will be affected by **Include Namespace**.

- Advanced Checkpoint with Namespace
- Non-volatile Memory Dynamic
- Non-volatile Memory Dynamic Input
- Non-volatile Memory Dynamic with Default
- Read-only Parameter Input with Namespace

User Interface

Scope

Scope

Item	Behavior when selected
Current module	Only the current `SCS module/Vector-Based Screen Editor repository/PLC Open XML file/C file` will be included in the search.
Current project	The currently opened project will be included in the search.
PLUS+1 GUIDE projects in file structure	All PLUS+1® GUIDE projects, packed and unpacked, at the specified file path (including sub-folders) will be included in the search. PLUS+1® GUIDE files other than `.p1x` and `.p1p` that do not belong to a project will not be included.

Searchable Items Tree

Use the **Searchable Items** tree to select which items to include in a search. At root level, each branch represents a category of searchable items. All searchable items in a root level branch will find the same type of data.

Component
Use this branch to search for components.

Branch/Item	Matches against
Parameter Value	One of the parameter values of a component, see the chapter *Parameter value* on page 83.
Component Name	The name of a component
Output Pin Datatype	The data type of one of the output pins of a component

Parameter value

Use this branch to search for parameter values. Items in **Application Interface**, Callable Components and **Constants** will match the search string with the first parameter value of specific types of components. **Other Parameter Values** will match the search string with any parameter value that would not be found by the items in **Application Interface**, **Callable Components** or **Constants**.

Use **Other Parameter Values** to find additional values of components, such as the access level of a checkpoint or the comment of a Call Module component.

Application Interface

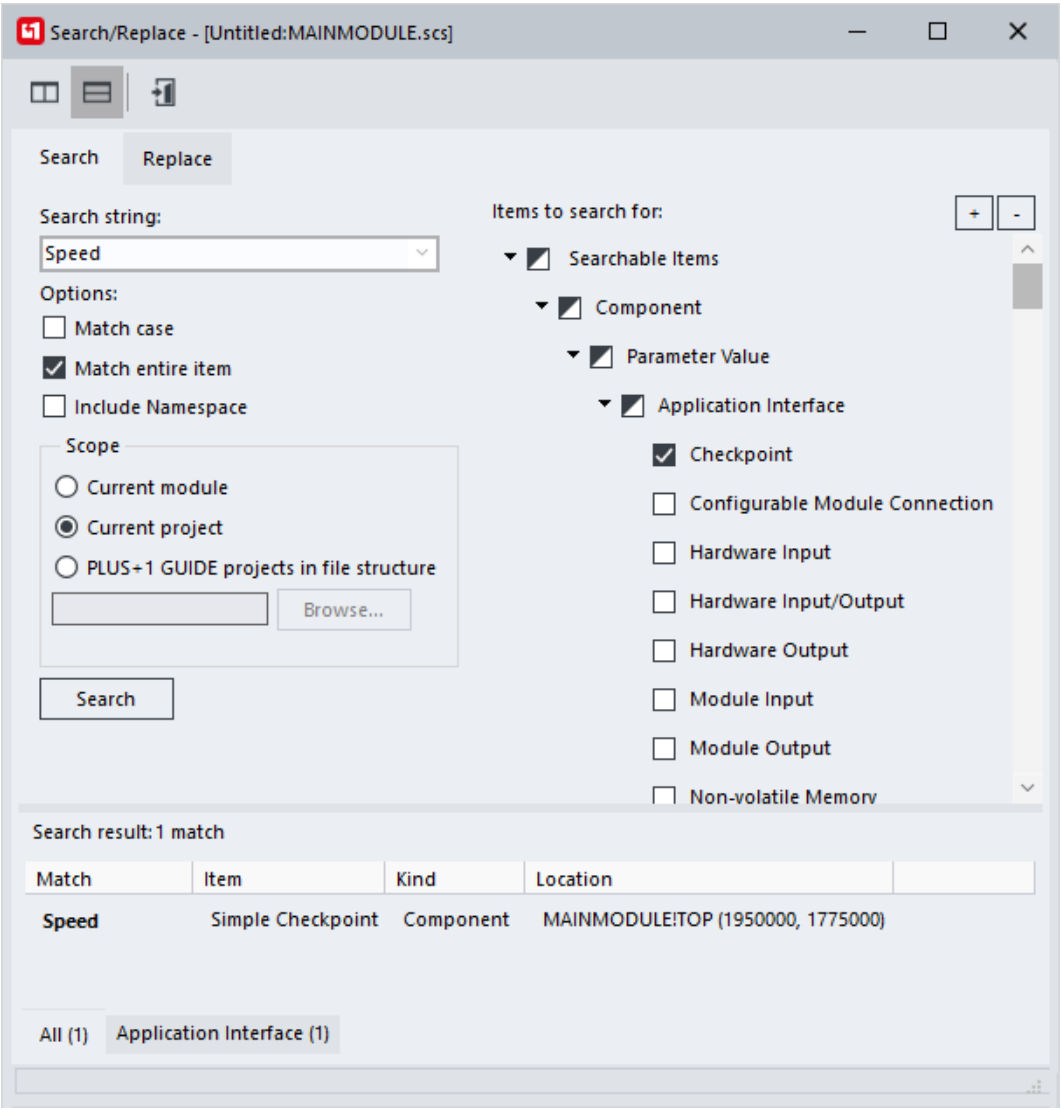
Use this branch to search for application interface signals. The values of the components will be matched against the search string.

Item	Components included when searching
Checkpoint	Simple Checkpoint, Advanced Checkpoint, Advanced Checkpoint with Namespace
Configurable Module Connection	Configurable Module Connection
Hardware Input	Hardware Input, Hardware Input Typed
Hardware Input/Output	Hardware Input/Output, HardwareInput/Output Typed, Bios In/out Replaced
Hardware Output	Hardware Output, Initialize Hardware Output
Module Input	Module Input, Module Input Typed, Module Bus Input
Module Output	Module Output, Module Bus Output
Non-volatile Memory	Non-volatile Memory Input/Output, Non-volatile Memory Input Typed, Non-volatile Memory Input, Non-volatile Memory Dynamic with Default, Non-volatile Memory Dynamic Input, Non-volatile Memory Dynamic
Read Output from Hardware	Read Output from Hardware, Read Output from Hardware Typed
Set Pulse	Set Pulse
Set Value	Set Value

User Interface

Search for Checkpoint Speed

Example: **Search result:** > **Speed** component



1. Enter `Speed` as **Search string**.
2. Select **Match entire item**.

   Since the exact **Search string** is known, selecting **Match entire item** will exclude other possible matches such as **Speedlimit** or **MaxSpeed**.
3. In the **Searchable Items** tree, select **Application Interface** > **Checkpoint**.
4. Select **Scope** > **Current project** .
5. Select **Search**.
6. See the **Search result** view, it is updated with one search match: **Speed**.
7. Click **Search hit** to navigate to checkpoint **Speed**.

Callable Components

Use this branch to search for components calling other entities. The identifier of the entity to call will be matched against the search string.

User Interface

Item	Components included when searching
Call Method of Externally Defined Class	Call Method Of Externally Defined Class
Call Module	Call Module
Call POU	Call POU
Show Screen	Show Screen
Write Applog	Write Applog

Constants

Use this branch to search for pages containing a traceability property. The value of the component will be matched against the search string.

Item	Components included when searching
Array Constant from File	Array Constant from File For this component, the name of the array will be matched against the search string.
Autotyped Constant	3 Digit Autotype, 6 Digit Autotype
Boolean Constant*	False, True
Null*	Null
Time Base	Time Base
Typed Constant	3 Character, 6 Character, Multi-character
Zero*	Zero

* The components for the corresponding item do not have an editable value. These components will be matched against the values in the following table. (False and True can be matched against two different values.).

Component	Matching values
False	False, F
True	True, T
Null	Null
Zero	0

Texts and Ports

Use this branch to search for bus ports, page ports, and CAD texts.

Item	Matches against
Bus Port	Port name
Page Port	Port name
Text	Text content
Multi-line text box	Text content

Page Properties

Use this branch to search for pages.

Item	Matched page property
Function Name	Function Name
Function Version	Function Version
Is Object	Object
Library	Library
LinkID	LinkID

User Interface

Item	Matched page property
Namespace	Namespace The value of the property will be matched; search option **Include Namespace** will not affect the search for this item.
Page ID	Page ID
Page Name	Page Name
Page View Access	Page View Access
View License	View License

Test Definitions

Use this branch to search for pages containing a test definition.

Item	Matches against
Test Definition	Test ID of a test definition referenced by a page.
Test Definition in List	Test ID of a test definition located in a test definition of List type that is referenced by a page. For example, consider a test definition of List type called MyTestList containing a test definition of Table type called MyTestTable. This search item would find MyTestTable and the search result hyperlink would target the page where MyTestList is referenced.

Traceability Properties

Use this branch to search for pages containing a traceability property.

Item	
Description	Description of a traceability property referenced by a page.
ID	ID of a traceability property referenced by a page.
Title	Title of a traceability property referenced by a page.

Screen Library

Use **Screen Library** items to search for definitions in the **Screen Library of the Vector-Based Screen Editor**.

Item	Included when searching
Image Definition Description	Description of an image definition.
Image Filename	Source filename of an image definition.
Image Format	Format of an image definition.
Image List Name	Name of an image list.
Text Definition Content	Content of a text definition. All translations will be included.
Text Definition Description	Description of a text definition.
Text List Name	Name of a text list.

Screen/Application Log Objects

Use **Screen/Application Log Objects** to search for objects in **Screen/Application Log Definitions**. Searchable items are divided into four sub-branches depending on the kind of data that is matched against the Search string.

Screen/Application Log Objects by Name

Use this branch to match the search string against screen object names.

Item	Included when searching
Applog Text Object	Name of an **Applog Text List Object**
Applog Text List Object	Name of an **Applog Text Object**

User Interface

Item	Included when searching
Hardware Image Object	Name of a **Hardware Image Object**
Image List Object	Name of an **Image List Object**
Image Object	Name of an **Image Object**
Line Object	Name of a **Line Object**
Text List Object	Name of a **Text List Object**
Text Object	Name of a **Text Object**
Touch Area Object	Name of a **Touch Area Object**
Widget	Name of a **Widget**

Screen/Application Log Objects by Definition

Use this branch to match the search string against properties of screen library definitions that are referenced by screen objects.

Item	Included when searching
Image Definition Description	Description of an image definition.
Image Filename	Source file of an image definition.
Image Format	Format of an image definition.
Image List Name	Name of an image list.
Screen Definition Name	Name of a widget definition
Text Definition Content	Content of a text definition. All translations will be included.
Text Definition Description	Description of a text definition.
Text List Name	Name of a text list.

Font

Use this branch to find texts and text lists that uses a certain font.

Item	Included when searching
Font Attributes	Attributes of a font (bold and italics); search string shall be set to **B** for bold, **I** for italics, or **BI** for both bold and italics.
Font Name	Name of a font.
Font Size	Size of a font.

Library Widget

Use this branch to search for library widgets.

Item	Included when searching
Library Description	Description of a widget library.
Widget Id	Id of a library widget.
Widget Version	Version of a library widget.

Interface Signals

Use this branch to search for screen objects connected to a certain interface signal.

Item	Included when searching
Interface Signal Data Type	Data type of an interface signal.
Interface Signal Name	Name of an interface signal.

Screen/Applog Definitions

Use this branch to search for screen or application log definitions.

User Interface

Item	Included when searching
Application Log Definition	Name of an application log definition
Application Log Definition by Interface Signal Data Type	Data type of an interface signal in an application log definition.
Application Log Definition by Interface Signal Name	Name of an interface signal in an application log definition.
Screen Definition	Name of a screen definition.
Screen Definition by Interface Signal Data Type	Data type of an interface signal in a screen definition.
Screen Definition by Interface Signal Name	Name of an interface signal in a screen definition.
Screen Definition by POU Call Name	Name of a POU call within a screen definition.
Screen Definition by POU Name	Name of a POU being called from within a screen definition.

PLC

Use this branch to search for contents in PLC Open XML files.

Item	Matches against
Graphical Implementation	Element property in a function block diagram, sequential function chart or ladder logic POU.
PLC Data Type	Contents of a PLC data type line.
PLC Global Variable	Contents of a PLC global variable line.
PLC Unit	Name of a POU, data type or global variable.
POU Interface	Contents of a POU interface line.
Textual Implementation	Contents of a C, structured text or instruction list implementation line.

C Code File

Use this branch to perform a free text search in a C file.

Item	Included when searching
C Header Line	Lines in an included C header file.
C Source Line	Lines in an included C source file.

Project Properties

Use this branch to search for project properties. Unless explicitly stated, the values of the searchable items can be found in the **Project Manager**.

Item	Comment
API Document	API Specification name
Application Name	Property of the Application node
Application Parameters	Property of the Application node
Architecture Document	Only available if Architecture export has been enabled
Array Comment	Property of added Array Constant from File
Array File Name	Property of added Array Constant from File
Array Parameters	Property of added Array Constant from File
C Code File	Name of added C code file
Compiled Code Package Comment	Property of added Compiled Code Package
Compiled Code Package File Name	Property of added Compiled Code Package
Compiled Code Package Interface Name	Property of added Compiled Code Package
Compiled Code Package Parameters	Property of added Compiled Code Package

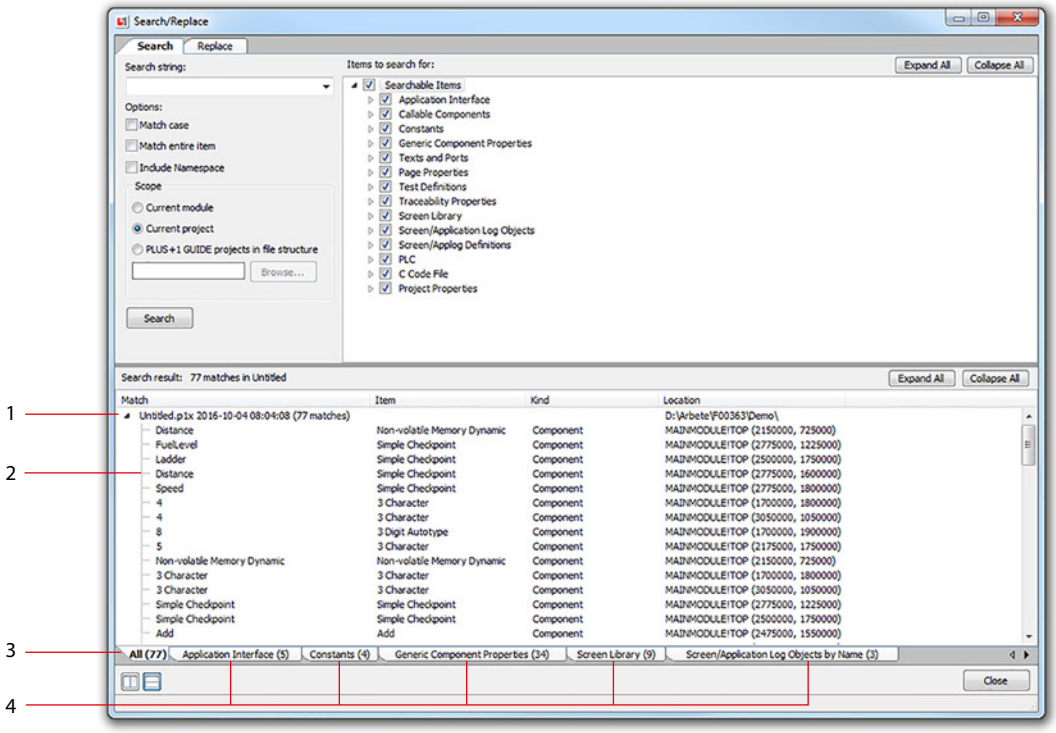
User Interface

Item	Comment
Compiled Code Package Type	Property of added Compiled Code Package
Document Comment	Property of added Document
Document File Name	Property of added Document
Document Parameters	Property of added Document
GUI Version	Version of PLUS+1® GUIDE that made the most recent change to the project. Not shown in Project Manager.
Hardware Comment	Property of the Hardware Description node
Hardware Description	Property of the Hardware Description node
Hardware Part Number	Property of the Hardware Description node
LHX Readme Comment	Property of added LHX Readme File
LHX Readme File	Property of added LHX Readme File
LHX Readme Parameters	Property of added LHX Readme File
Main Module Comment	Property of the main module
Main Module File Name	Property of the main module
Main Module Name	Property of the main module
Main Module Parameters	Property of the main module
Module Comment	Property of an added module
Module File Name	Property of an added module
Module Name	Property of an added module
Module Parameters	Property of an added module
PID Description	Property of Application ID
PID System Id	Property of Application ID
PID Type	Property of Application ID
PID Version	Property of Application ID
PLC Open XML File	The name of an added PLC Open XML File
Plugin Description	Property of an added External Kernel Function
Plugin File Name	Property of an added External Kernel Function
Plugin Interface	Property of an added External Kernel Function
Plugin Socket Name	Property of an added External Kernel Function
Plugin Type Project Name	Property of an added External Kernel Function
Range Definitions File Name	Name of an added Global Range file
Read Only Parameter Comment	Property of an added Read-only Parameter file
Read Only Parameter File Name	Property of an added Read-only Parameter file
Read Only Parameter Port	Property of an added Read-only Parameter file
Read Only Parameter Type	Property of an added Read-only Parameter file
Sys File Name	Property of the System File (kernel)
Sys File Parameters	Property of the System File (kernel)
Target Part Number	Property of the Output file
Target Serial Number	Property of the Output file
Test Case Definition File Name	Name of an added Test Definition file
Test Status File Name	Name of the Test Status file
Tool Key Configuration	Properties of an added Tool Key
Tool Key Parameters	Properties of an added Tool Key

Project property search hits differ from other search hits. While other search hits target an item inside a PLUS+1® GUIDE project, a project property search hit targets a PLUS+1® GUIDE project. Searches for project properties can be useful when search scope is set to **PLUS+1 GUIDE projects in file structure**.

User Interface

Search Result View



Search result view

Item	Name	Description
1	Project with search results	Each project containing search results gets its own row in the search result view. Project name, time and date of the most recent save and the file system path to the project is displayed. Clicking the row will open the project
2	Search result	Each search result row represents a search hit. Clicking a search result row moves the PLUS+1® GUIDE view to the search hit, opening any editors if needed.
3	General search result tab	All search hits of the most recent search are displayed in this tab. Each search hit is presented with general information about it. See *General Search Result Tab* on page 91 for more information
4	Specific search result tabs	One specific search result tab is created for each branch in the searchable items tree that produces at least one search hit. Specific search result tabs present search hit data in a non-general way suitable for the type of search hit it presents. See *Specific Search Result Tab* on page 91 for more information

User Interface

Search result view context menu

Access the context menu by right-clicking the search result view.

Item	Description
Copy hyperlink	Copy the hyperlink of the current row to the clipboard
Copy path to folder	Copy the file system path of the project to which the current row belongs to the clipboard
Export search result tab to CSV	Export the current tab to a .csv file

General Search Result Tab

Use general search result tab to get an overview of and brief information about all search results.

Column	Description
Match	Contains the text of the searchable item that was matched against the search string with the matching part bold. If no search string was specified, no part of the text will be bold
Item	A general description of the search hit
Kind	A general categorization of the search hit
Location	A readable representation of the query part of the search hit hyperlink. When hovering a search hit row, its hyperlink in its entirety is shown in the status bar of the search dialog

Specific Search Result Tab

Use specific search result tabs to view detailed information about each search result and to group search results by category. Each specific tab has the **Match** and **Location** columns as described in *General Search Result Tab* on page 91. These columns will not be described in this topic since they have the same meaning and positioning. (**Match** is always the leftmost column, **Location** is always the rightmost column).

Application Interface, Callable Components, Constants, Generic Component Properties

These search result tabs have a dynamic number of columns. If a component does not have an output type or a parameter value to display, the corresponding column is empty.

Column	Description
Component	Component name.
Output type n	Output type of the n:th output. Set to the empty string if there is no output type for the n:th pin or if the n:th pin does not exist. n = 1..maximum number of outputs for a component in the tab.
Value n	The n:th value of the component. Set to the empty string if the component has less than n values. n = 1..maximum number of values for a component in the tab.

User Interface

Example—Components in a Search Result

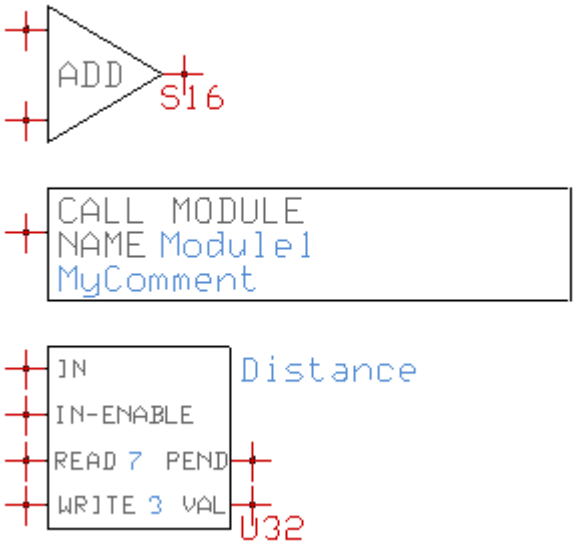
PLUS+1® GUIDE code components

- **Add** component has output type S16 but no values.
- **Call Module** component does not have output type, instead it has values defining the module to be called and the comment.
- **Non-volatile Memory Dynamic** component.

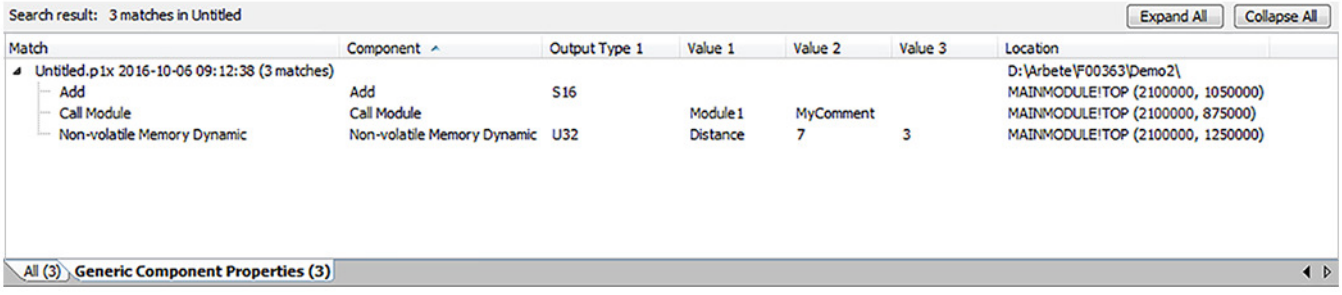
Search result view components

- **Output Type 1** displays output type.
- **Value 1** displays name.
- **Value 2** displays read access level.
- **Value 3** displays write access level.

PLUS+1® GUIDE code



Search result view



Search result: 3 matches in Untitled — Expand All — Collapse All

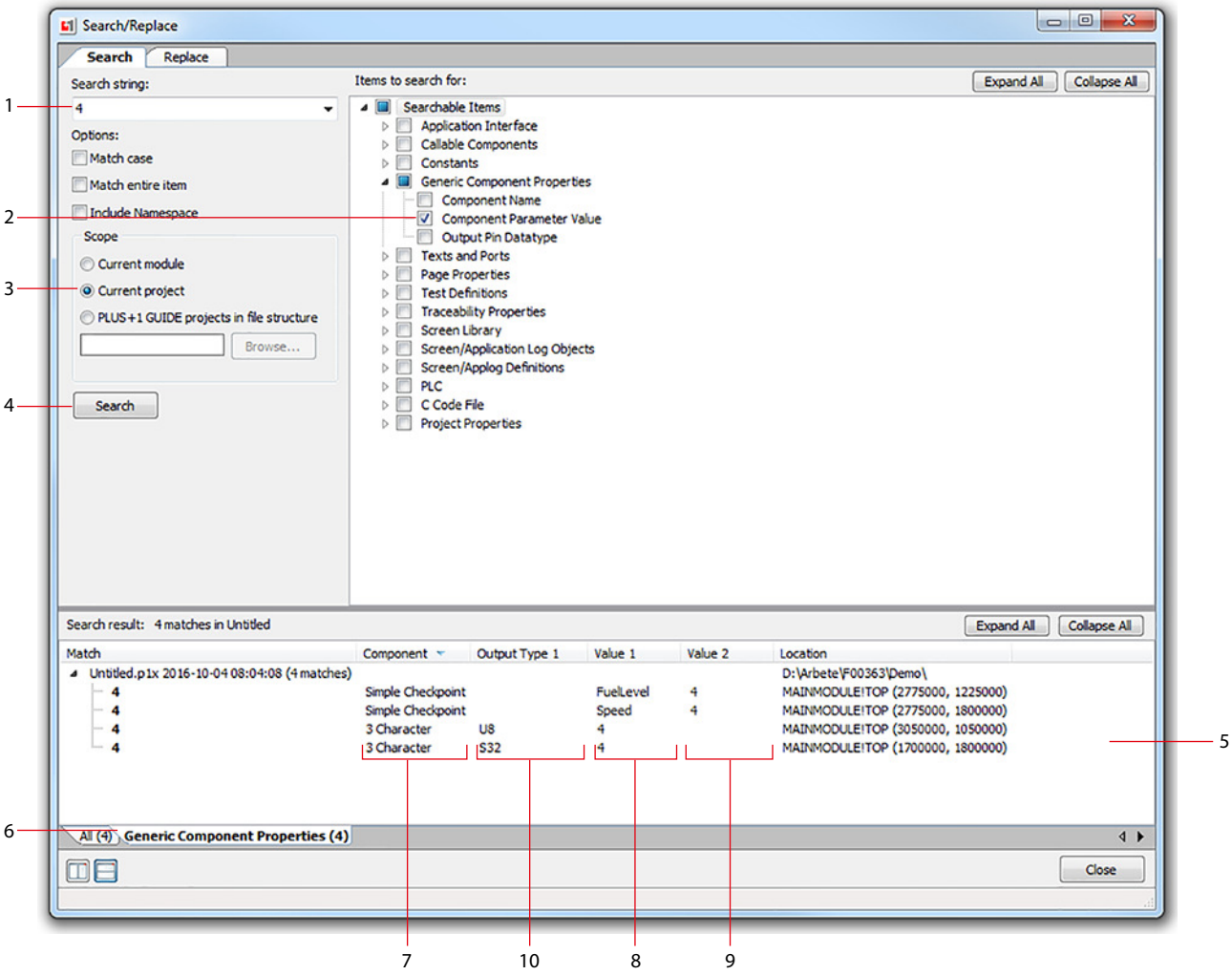
Match	Component	Output Type 1	Value 1	Value 2	Value 3	Location
Untitled.p1x 2016-10-06 09:12:38 (3 matches)						D:\Arbete\F00363\Demo2\
Add	Add	S16				MAINMODULE!TOP (2100000, 1050000)
Call Module	Call Module		Module1	MyComment		MAINMODULE!TOP (2100000, 875000)
Non-volatile Memory Dynamic	Non-volatile Memory Dynamic	U32	Distance	7	3	MAINMODULE!TOP (2100000, 1250000)

All (3) | Generic Component Properties (3)

User Interface

Example – Search for all checkpoints with access level 4

Access level is specified in the second component parameter value of a Simple Checkpoint. To find all checkpoints with access level 4, a generic component property search must be performed.



1. Enter 4 as **Search string**.
2. Select **Generic Component Properties** > **Component Parameter Value** in the **Searchable Items tree**.
3. Set **Scope** to **Current Project**
4. Press **Search**.
5. See the **Search result** view, it is updated with four search matches.
6. Switch to specific search result tab **Generic Component Properties**, examine the matches in detail. Among the matches there are two checkpoints and two constants.
7. Click column header **Component** twice to sort the search results in reverse alphabetic order. Now the two checkpoints appear at the top.
8. See **Value 1**, the names. **Simple Checkpoints** are **FuelLevel** and **Speed**.
9. See **Value 2**, access level, there is 4 for both.
10. See **Output Type 1**, output types are displayed.

User Interface

Texts and Ports

Column	Description
Item Type	Type of search hit. Possible types include Text, Page Port Bus, Page Port Wire, Bus Port Bus, and Bus Port Wire.

Page Properties

Column	Description
Property	Name of matched page property
Page	Page name
References	Number of other pages linked to the page of the matching page property if that page is referenced. If the page of the matching page property is not referenced the value will be zero. This is the same value as References in the Edit Page dialog.

Test Definitions

Column	Description
Test Definition	Test definition name
Parent List	Name of list containing the matched test definition or the empty string if the matched test definition is not located in a list.

Traceability Properties

Column	Description
Traceability Property	Type of traceability property. Possible types include ID, Title, and Description.
Page	Page where the traceability property was found. Since search hit hyperlinks indicates the position of the page in its parent page, this column will show one more level of the page hierarchy than Location.

Screen Library

Column	Description
Property	Matched property of the search hit. For example, if a text definition is found by matching its description, this column would be set to Description.
Definition Type	Type of search hit. For example, if a text definition is found, this column would be set to Text.

Screen/Application Log Objects by Name, Screen/Application Log Objects by Screen Library Definition

Column	Description
Object Name	Object name shown in the Screen Manager
Object Type	Type of object as shown in the Selector. Applog objects get the prefix “Applog"

Font

Column	Description
Name	Font name
Size	Font size
Attributes	Font attributes. Bold is indicated by B, italic is indicated by I, both bold and italic is indicated by BI.

User Interface

Interface Signals

Column	Description
Name	Interface signal name
Data Type	Interface signal data type
Definition	Definition Screen/application log definition where the signal is defined

Screen/Applog Definitions

Column	Description
Matched Net	If an Interface Signal Name or an Interface Signal Datatype was matched, that interface signal is displayed in this column

PLC

Column	Description
Item	Description of search hit. In case of FBD or LD implementation, the value of the element property is shown in this column. Otherwise, the search hit is described according to the following format: [PLCOpen XML file name]/[POU/global variable/data type name]:[line number] For example, if the second line in a structured text implementation called CruiseControl located in PLC Open XML file PlcOpen.xml is found, this column would have the following value: PlcOpen/CruiseControl:2
Item Type	Search hit type. This column corresponds to the items in the PLC branch in Searchable Items tree.

C Code file

Column	Description
Line	Line number
File Type	Type of file. Set to C Source (.c files) or C Header (.h files).

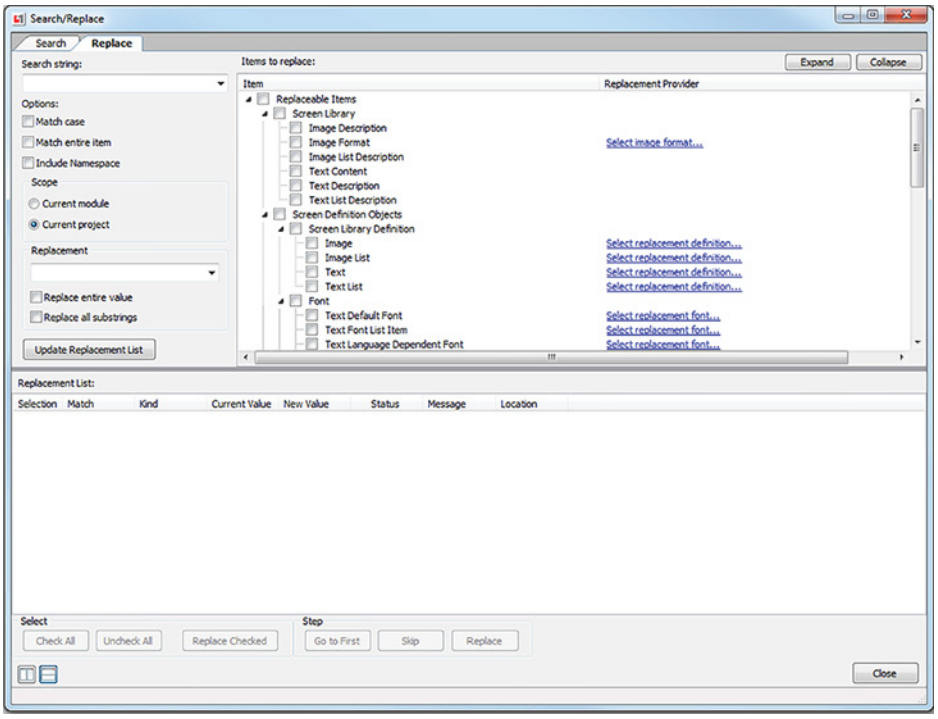
Project Properties

Column	Description
Property	Type of project property

User Interface

Replace

Use replace functionality to replace items in the currently opened project. The first step is to specify search criteria and replacement data. Then the **Update Replacement List** button has to be pressed in order to generate a replacement list containing all items found using the specified search criteria. For each item in the replacement list it is shown how the replacement data would be applied to that item and whether the replacement would be valid or not. Replacements are performed via the replacement list, either one item at a time or multiple items at once. Each replacement has a hyperlink so that each replacement item can be inspected before performing the replacement.



Replace dialog overview

Name	Description
Search string	Enter the text to search for. If no text is specified, all searchable items will be found.
Options	Determine how Search string will be matched against replaceable items. See *Options* on page 97 for a detailed description.
Scope	Define the scope to search in. See *Scope* on page 97 for a detailed description
Replaceable Items tree	Select which replaceable items to include. See *Replaceable Items tree* on page 97 for a detailed description
Replacement provider	Select replacement from a dialog. See *Replacement Provider* on page 100 for a detailed description.
Replacement settings	Specify how the replacement will be performed. See *Replacement settings* on page 100 for a detailed description.
Update replacement list button	Find all items according to the criteria in [1], [2], [3], and [4]; apply replacements according to [6]; populate [8]. A progress dialog is shown if these operations will be time-consuming.
Replacement list	View and perform replacements. See *Replacement List* on page 101 for a detailed description.
Alignment buttons	Align the two main components of this dialog horizontally or vertically.
Close button	Close the search/replace dialog.

User Interface

Options

Item	Behavior when checked
Match case	Search for replaceable items will be case sensitive. For example, if this option is checked a search for component name "Add" will find Add components while a search for component name "add" will not find Add components. If this option is unchecked, a search for "add" will find Add components.
Match entire item	The entire value of the replaceable item will be matched against Search string. For example, a search for component name "Add" will find both Add and add Add Capped components if this option is unchecked; if it is checked only Add components will be found.
Include Namespace	When searching for components with Namespace, the entire value including Namespace will be matched against Search string. This option is ignored in all other cases.

Scope

Options

Item	Behavior when selected
Current module	Only the current `SCS module/Vector-Based Screen Editor repository/PLC Open XML file/C file` will be included in the search for replacements.
Current project	The currently opened project will be included in the search for replacements.

Replaceable Items tree

Screen Library

Replace properties of screen library definitions.

Item	What to replace
Image Definition Description	Description of an image definition
Image format	Format of an image definition
Image List Description	Description of an image list
Text Definition Content	Content of a text definition. All translations will be included
Text Definition Description	Description of a text definition
Text List Description	Description of a text list

Screen Definition Objects

Replace properties of screen definitions objects. This branch is divided into sub-branches depending on the type of property to replace.

Replace screen library definitions referenced by screen definition objects.

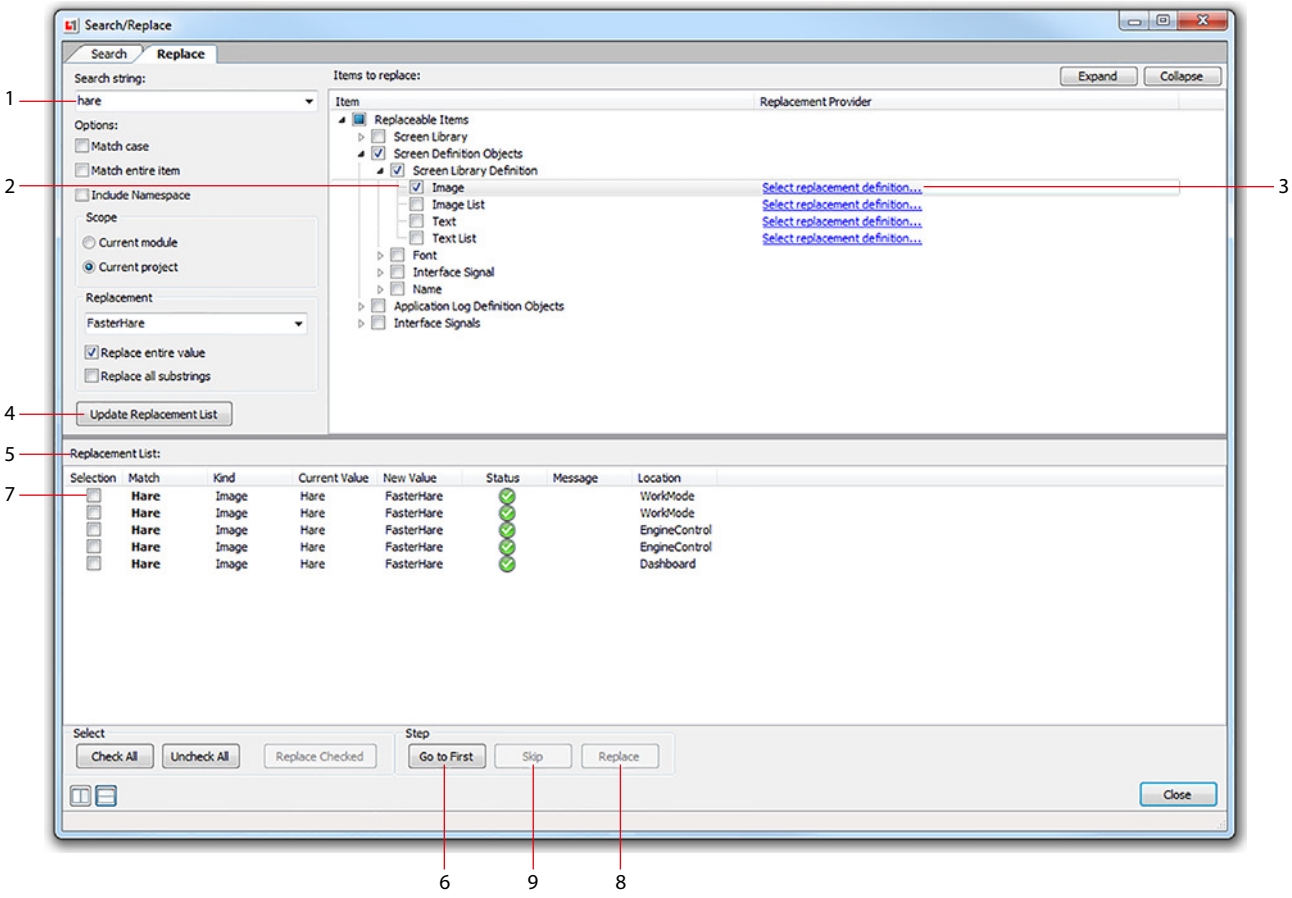
Screen Library Definition

Item	What to replace
Image	Image definition
Image List	Image list definition
Text	Text definition
Text List	Text List definition

Example—Replace a Proper Subset of All Instances of an Image Definition

In this example, an image depicting a hare has been used to indicate high speed. Now a second hare image has been added that will indicate an even higher speed. Some instances of the original hare image will be replaced and some will be kept. In order to accomplish this, the following procedure is followed:

User Interface



The number in parentheses refer to the image callout.

1. Enter **Search string** `hare` (1).
2. Select to replace image definitions referenced by screen objects (2).
3. Use the **Replacement Provider** (3) to select one of the existing image definitions in the project, in this case an image definition with the description FasterHare
4. Press **Update Replacement List** (4) to generate the **Replacement List** (5).
5. Press **Go to First** (6) to select the first row in the **Replacement List** (7) and to navigate to the screen object it represents inside Vector-Based Screen Editor. Assume that the image definition of this screen object shall be replaced.
6. Press button **Replace** (8) to replace it and go to the next row in the **Replacement List**.
   The second row will be selected and Vector-Based Screen Editor will navigate to the corresponding screen object. This time, assume that the image is fine as it is.
7. Press button **Skip** (9) to move to the next row in the **Replacement List**.
   Next row will be selected and Vector-Based Screen editor will navigate to the corresponding screen object
8. Use **Replace** to traverse the rest of the **Replacement List**.
9. Use **Skip** for the same purpose as **Replace**.

Font

Replace fonts used by screen definition objects.

User Interface

Item	What to replace
Text Default Font	Default font of a text. This is the font that will be displayed unless advanced font features like **Font List** or **Language Dependent Font** are used
Text Font List Item	Font list item of a text
Text Language Dependent Font	Language dependent font of a text
Text List Default Font	Default font of a text list. This is the font that will be displayed unless advanced font features like **Font List** or **Language Dependent Font** are used
Text List Font List Item	Font list item of a text list
Text List Language Dependent Font	Language dependent font of a text list

Interface Signal

Replace interface signals used by screen definition objects.

Item	What to replace
Hardware Image	Signal connected to a hardware image.
Image	Signal connected to an image.
Image List	Signal connected to an image list.
Line	Signal connected to a line.
Text	Signal connected to a text.
Text List	Signal connected to a text list.
Touch Area	Signal connected to a touch area.

Name

Replace screen definition object names.

Item	What to replace
Hardware Image	Name property of a hardware image.
Image	Name property of an image.
Image List	Name property of an image list.
Line	Name property of a line.
Text	Name property of a text.
Text List	Name property of a text list.
Touch Area	Name property of a touch area.

User Interface

Application log definition objects

Replace properties of application log definitions objects. This branch is divided into sub-branches depending on the type of property to replace.

Screen Library Definition

Replace screen library definitions referenced by screen definition objects.

Item	What to replace
Text	Text definition
Text List	Text List definition

Interface Signal

Replace interface signals used by application log definition objects.

Item	What to replace
Text	Signal connected to a text.
Text List	Signal connected to a text list.

Name

Replace application log definition object names.

Item	What to replace
Text	Name property of a text.
Text List	Name property of a text list.

Interface signals

Replace properties of interface signals. For both items in this branch, the signal name will be used to search for replacements.

Item	What to replace
Signal name	Signal name.
Signal data type	Signal data type.

Replacement Provider

Each replaceable item may have a replacement provider, indicated by a blue, underlined text. By clicking the text a dialog is opened. Use the dialog to select a replacement value. When button OK is clicked in the dialog, the selected value will be set as the replacement.

All replacement values are specified as texts. By using replacement providers, it is ensured that the replacement value is correctly specified. Therefore it is recommended to use replacement providers when they are available.

Replacement settings

Replacement value is provided by replacement providers or as free text.

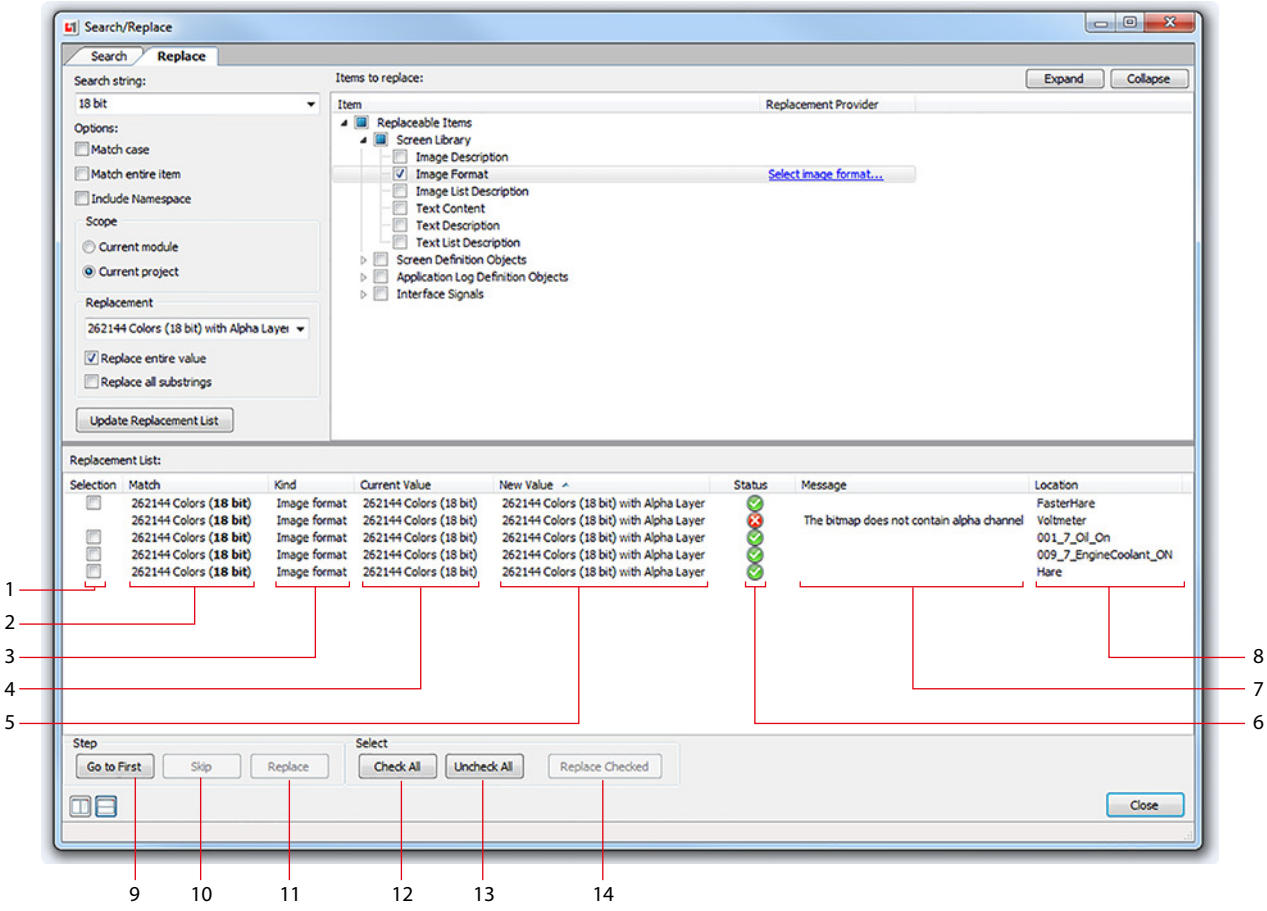
User Interface

Replacement options

Item	Behavior when selected
Replace entire value*	Regardless of the **Search string**, the entire value of the replaceable item will be replaced with the replacement value. For example, if **Search string** is set to `18 bit`, **Replacement** is set to `Monochrome (1 bit)`, **Replace entire value** is checked, and the item to replace has the value `262144 Colors (18 bit)`; then the value resulting from the replacement would be `Monochrome (1 bit)`.
Replace all substrings*	All instances of the **Search string** inside the replaceable item will be replaced. For example, if **Search string** is set to `de`, **Replacement** is set to `a`, **Replace all substrings** is checked, and the item to replace has the value `dependent`; then the value resulting from the replacement would be `apenant`.

* Replace entire value and Replace all substrings are mutually exclusive.

Replacement List



Replacements can be performed by selecting items in the list or by stepping through the list of replacements one at a time.

Replacement List

Item	Column name	Description
1	Selection	Checkbox used for selecting a row
2	Match	The value that was matched against Search string
3	Kind	Classification of the replaceable item

User Interface

Replacement List (continued)

Item	Column name	Description
4	Current Value	Value that would be replaced
5	New Value	Value that would replace Current Value
6	Status	Indicate whether the replacement is valid or not
7	Message	If the replacement is not valid, an error message may be displayed in this column
8	Location	Location of the replaceable item. Click a row in this column to navigate to the replacement

Step buttons

Item	Button name	Behavior when selected
9	Go to First	Select the first valid row in the replacement list and navigate to the corresponding item
10	Skip	Select next valid row after the selected row in the replacement list and navigate to the corresponding item
11	Replace	Replace the item at the currently selected row in the replacement list, then select next valid row after the selected row and navigate to the corresponding item.

Select buttons

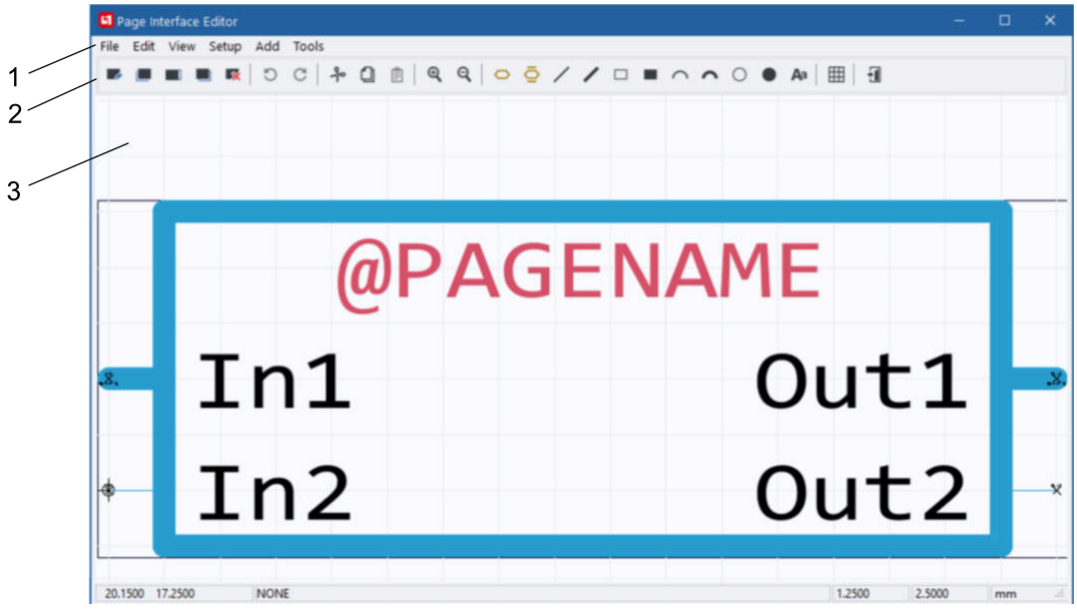
Item	Button name	Behavior when selected
12	Check All	Check all valid items in the replacement list
13	Uncheck All	Uncheck all items in the replacement list
14	Replace Checked	Replace all checked items in the replacement list

User Interface

Page Interface Editor

Toolbar > or

Add > **Page Interface Editor**



Page Interface Editor Window Description

Item	Description
1	Menu, use the menu bar to access **Page Interface Editor** window commands.
2	Toolbar, use the toolbar to access commonly used **Page Interface Editor** window commands.
3	Drawing area, edit the page top view here.

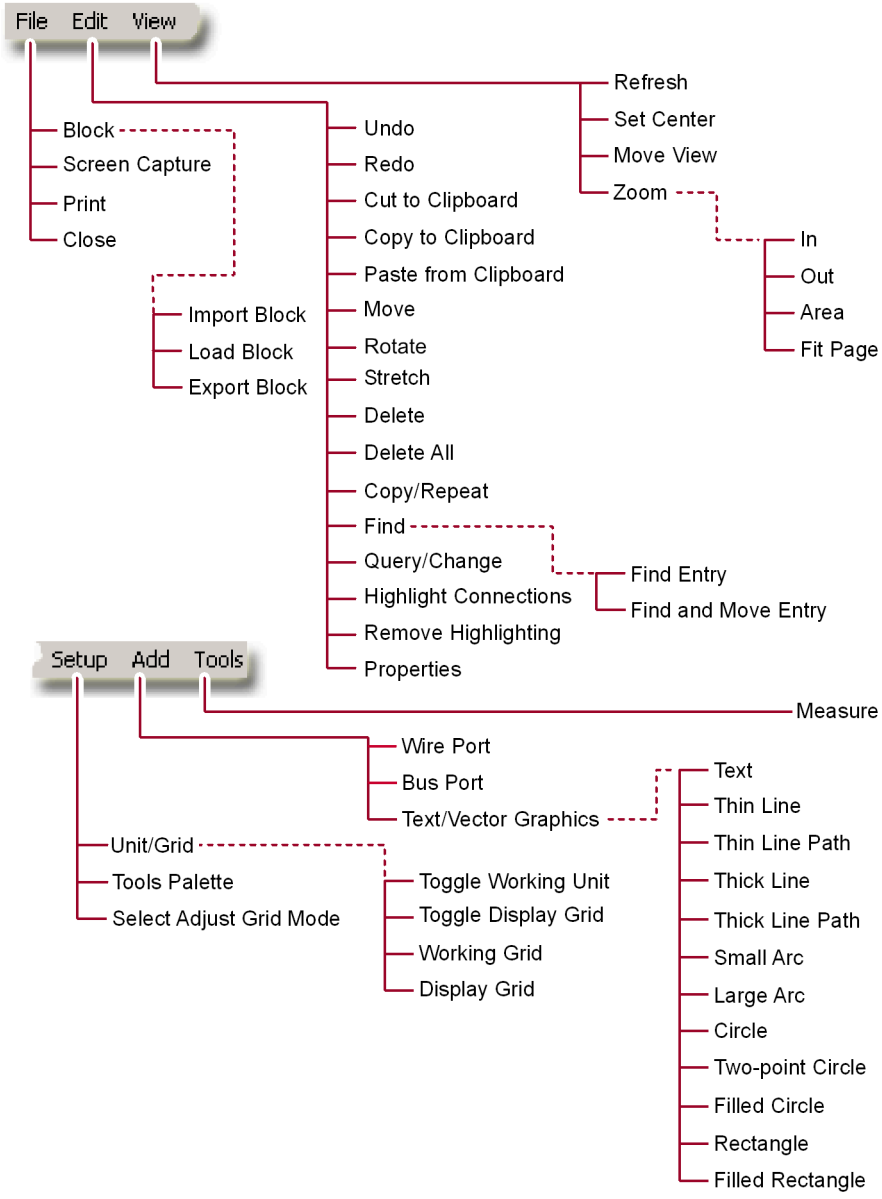
Use the **Page Interface Editor** window to edit the "top view" of a page.

Leave the **Layer** value of the **@PAGENAME** placeholder set to the Layer value of PageName. Changing this value may cause the page name to disappear. It can also cause compile problems and problems with the advanced features of PLUS+1® GUIDE.

For more information, see *About Pages, Page Top views, and the Page Interface Editor Window* on page 112.

User Interface

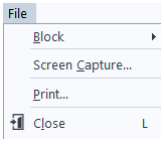
Page Interface Editor Window Menus



Use the menu bar to access **Page Interface Editor** window commands. Buttons in this window toolbar duplicate commonly used commands. See *Page Interface Editor Window Toolbar* for more about these buttons.

User Interface

Page Interface Editor—File Menu



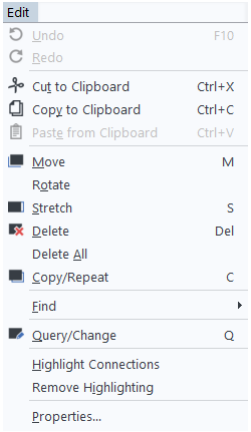
The commands in this menu manage **Page Interface Editor** window files.

File menu description

Item	Description
Block	• **Import Block**—displays the **Select Import Preview** window, select a **DXF ASCII** format file to import into Page Interface Editor window's Drawing Area. • **Load Block**—displays the **Schematics Symbol Editor** window, select a SCS format file. Any top view items in the file (such as entries and graphics) in the file import into the Page Interface Editor window's Drawing Area. • **Export Block**—displays the **Symbol Block Binary Export** window after you select items to export from the Page Interface Editor window's Drawing Area. Use the **Symbol Block Binary Export** window to export the selected items to an SCS file.
Screen Capture	Displays the **Select Screen Dump Format** window after you select items in the Page Interface Editor window's Drawing Area. Use this window to choose to print the selected items, save them to a BMP or TIFF file, or copy them to the clipboard in a BMP or Metafile format.
Print	Displays the **Print** window, print the contents of the Page Interface Editor window's Drawing Area.
Close	Closes the Page Interface Editor window.

User Interface

Page Interface Editor—Edit Menu



The commands in this menu make changes to items in the **Page Interface Editor** window's Drawing Area.

Edit menu description

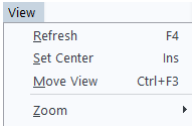
Item	Description
Undo	Reverses programming actions. Depending on computer memory, you can undo up to ten actions.
Redo	Reverses **Undo** commands. Depending on computer memory, you can reverse up to ten Undo commands. The **Options** window enables and disables the undo/redo function. To display this window, click **Options** in the **Setup** menu.
Cut to Clipboard	Deletes items selected in the Page Interface Editor window's Drawing Area and copies them to the Clipboard.
Copy to Clipboard	Copies items selected in the Page Interface Editor window's Drawing Area to the Clipboard.
Paste from Clipboard	Pastes the contents of the Clipboard into the Page Interface Editor window's Drawing Area.
Move	Moves items selected in the Page Interface Editor window's Drawing Area.
Rotate	Rotate items selected in the Page Interface Editor window's Drawing Area.
Stretch	Stretches graphic items such as lines and arc by moving selected vertexes. Moves selected text and entry items.
Delete	Deletes items selected in the Page Interface Editor window's Drawing Area. Selected items turn white. • The **Attributes** window—displays when you select a single item or identical items. In the Attributes window, click **OK** to delete your selection. • The **Select Item Class** window—displays when you select different items, select the items that you want to delete. Asterisks (*) identify items that will be deleted; dashes (–) identify items that will not be deleted. In this window, click **OK** to delete your selections.
Delete All	Displays a **Question** window with a **Delete all items?** message. Click **Yes** to delete all items in the Page Interface Editor window's Drawing Area.
Copy/Repeat	Copies items selected in the Page Interface Editor window's Drawing Area. Selected items turn white. Click to place copied items. Press **Esc** to stop placing copied items. This command only works within the Page Interface Editor window's Drawing Area.
Find	• **Find Entry**—displays a **Select Entry** window that lists each **Entry** (port) in the Page Interface Editor window's Drawing Area. Click port name in the **Select Entry** window to zoom and center the Page Interface Editor window's Drawing Area on the selected port. • **Find and Move Entry**—displays a **Select Entry** window that lists each **Entry** (port) in the Page Interface Editor window's Drawing Area. Click a port name in the **Select Entry** window to snap the selected port to where you click in the Page Interface Editor window's Drawing Area.
Query/Change	Use to change the properties (such as text, text size, color, line thickness) of items in the Page Interface Editor window's Drawing Area. Click an item whose property you want to change. A dialog box appropriate for the selected item displays. Change the property in this window.
Highlight Connections	Highlights items selected in the Page Interface Editor window's Drawing Area.

User Interface

Edit menu description (continued)

Item	Description
Remove Highlighting	Remove highlighting from items in the Page Interface Editor window's Drawing Area
Properties	For advanced users only.

Page Interface Editor—View Menu



The commands in this menu change the view in the **Page Interface Editor** window's Drawing Area.

View menu description

Item	Description
Refresh	Refreshes the Page Interface Editor window's Drawing Area.
Set Center	Centers the Page Interface Editor window's Drawing Area on where you click the pointer.
Move View	Displays a movable, transparent rectangle with white borders. Click to center the Page Interface Editor window's Drawing Area within the borders of this rectangle.
Zoom	• **In**—zoom in and center the Page Interface Editor window's Drawing Area on where you click the pointer. • **Out**—zoom out and center the Page Interface Editor window's Drawing Area on where you click the pointer. • **Area**—zoom the Page Interface Editor window's Drawing Area on an area defined by two clicks of the pointer. • **Fit Page**—sizes the view to fit all items on the page into the Page Interface Editor window's Drawing Area.

User Interface

Page Interface Editor—Setup Menu

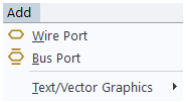
The commands in this menu set up the **Page Interface Editor** window's work environment.

Setup menu description

Item	Description
Unit/Grid	• **Toggle Working Unit**—click to toggle between using inches and millimeters as units of measurement. • **Toggle Display Grid**—turns the display grid off and on. The display grid is the grid that you see in the Drawing Areas of the PLUS+1 GUIDE, Page Interface Editor, and Module Viewer windows. • **Working Grid**—displays the **Define Work Grid** window, define the size (the distance between grid lines) of the working grid. Items in the Drawing Areas of the PLUS+1 GUIDE, Page Interface Editor, and Module Viewer windows snap to the working grid. This is an invisible grid. • **Display Grid**—displays the **Define Display Grid** window, define the spacing of the display grid. Press G to show a new display grid selection.
Tools Palette	Displays the **Icon Menu** window.
Select Adjust Grid Mode	Displays the **Setup Select Items Grid Adjust Mode** window, set the snap move behavior of items in the Page Interface Editor window's Drawing Area after you change to a new working grid with a different size. (Size is the distance between grid lines.) • **Always adjust reference to working grid**—moves an item from its position in the old grid in steps set by the size of the new grid. See the Example below. • **Ask to adjust single reference to working grid**—moving an item displays a **Question** window with the following text: **Force XY to grid? Yes/No**. — **Yes**—moves the item to an intersection in the new working grid. — **No**—moves the item from its position in the old grid in steps set by the size of the new grid. See the Example below. • **Adjust only multiple reference to working grid**—moves a single item to an intersection in the new working grid. Moves multiple items from their positions in the old grid in steps set by the size of the new grid. **Adjust only multiple reference to working grid** is the default setting. *The Example:* The size of the old grid is 0.25 mm; the size of the new grid is 1.0 mm. In the new grid, an item at 0.25 mm moves in 1.0 mm steps to 1.25 mm , 2.25 mm and 3.25 mm.

User Interface

Page Interface Editor—Add Menu

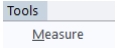


The commands in this menu add items to the **Page Interface Editor** window's Drawing Area.

Add menu

Item	Description
Wire Port	Displays the **Entry** window. Use this window to add a Wire Port to the Page Interface Editor window's Drawing Area. Enter the port name in **EntryName** field and click **OK**. Click to place the port in the Page Interface Editor window's Drawing Area.
Bus Port	Displays the **Entry** window. Use this window to add a Bus Port to the Page Interface Editor window's Drawing Area. Enter the port name in **EntryName** field and click **OK**. Click to place the port in the Page Interface Editor window's Drawing Area.
Text/Vector Graphics	Displays a drop-down list of text and graphics tools, add text and vector graphics to the Page Interface Editor window's Drawing Area.

Page Interface Editor—Tools Menu



The command in this menu measures distances.

Tools menu

Item	Description
Measure	Displays the distance between two points that you define by dragging the pointer.

User Interface

Page Interface Editor Window Toolbar

Buttons in the **Page Interface Editor** window toolbar access commonly used commands. See *Page Interface Editor Window Menus* on page 104 for a list of all commands available through the menu bar.

Page Interface Editor window toolbar

Page Interface Editor window toolbar description

Button		Description
	Query/Change	Use to change the properties (such as text, text size, color, line thickness) of items in the Page Interface Editor window's Drawing Area. Click an item whose property you want to change. A dialog box appropriate for the selected item displays. Change the property in this window.
	Move	Moves items selected in the Page Interface Editor window's Drawing Area.
	Stretch	Stretches graphic items such as lines and arcs by moving selected vertexes. Moves selected items.
	Copy/Repeat	Copies items selected in the Page Interface Editor window's Drawing Area. Selected items turn white. Click to place copied items. Press `to stop placing copied items. This command only works within the Page Interface Editor window's Drawing Area.
	Delete	Deletes items selected in the Page Interface Editor window's Drawing Area. Selected items turn white. The **Attributes** window displays when you select a single item or identical items. In the Attributes window, click **OK** to delete your selection. The **Select Item Class** window displays when you select different items. Use this window to select the items that you want to delete. Asterisks (*) identify items that will be deleted. dashes (–) identify items that will not be deleted. In the Select Item Class window, click **OK** to delete your selections.
	Undo	Reverses programming actions. Depending on computer memory, you can undo up to ten actions.
	Redo	Reverses Undo commands. Depending on computer memory, you can reverse up to ten Undo commands. The **Options** window enables and disables the undo/redo function. To display this window, click **Options** in the **Setup** menu.
	Copy to Clipboard and Delete	Deletes items selected in the Page Interface Editor window's Drawing Area and copies them to the Clipboard.
	Copy to Clipboard	Copies items selected in the Page Interface Editor window's Drawing Area to the Clipboard.
	Paste from Clipboard	Pastes the contents of the Clipboard into the Page Interface Editor window's Drawing Area.
	Zoom In	Zooms in and centers the Page Interface Editor window's Drawing Area on where you click the pointer.
	Zoom Out	Zooms out and centers the Page Interface Editor window's Drawing Area on where you click the pointer.
	Wire Port	Displays the **Entry** window. Use this window to add a Wire Port to the Page Interface Editor window's Drawing Area. Enter the port name in **EntryName** field and click **OK**. Then click to place the port in the Page Interface Editor window's Drawing Area.
	Port Bus	Displays the **Entry** window. Use this window to add a Wire Port to the Page Interface Editor window's Drawing Area. Enter the port name in **EntryName** field and click **OK**. Then click to place the port in the Page Interface Editor window's Drawing Area.
	Thin Line	Draws a thin blue line, typically used to show a single signal entering or leaving a page.
	Thick Line	Draws a thick blue line, typically used to either to show a signal bus entering or leaving a page or for the borders of a page.
	Hollow Rectangle	Draws a hollow rectangle, with thin borders.
	Filled Rectangle	Draws a filled rectangle.
	Small Arc	Draws an arc, with thin borders.

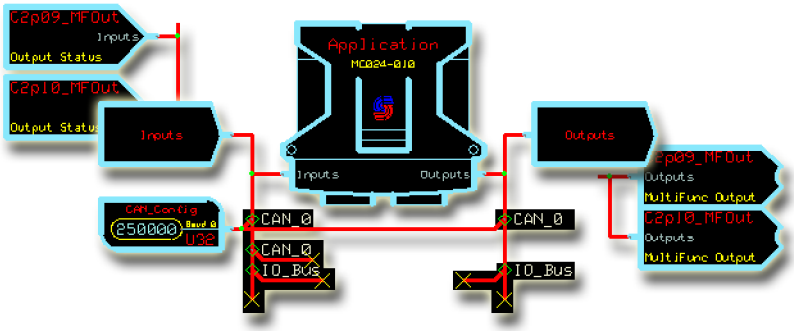
User Interface

Page Interface Editor window toolbar description (continued)

Button		Description
	Large Arc	Draws a hollow rectangle, with thin borders.
	Hollow Circle	Draws a hollow rectangle, with thin borders.
	Filled Circle	Draws a filled circle, with thin borders.
	Text	Displays a **Text Attributes** window after you click in the Page Interface Editor window's Drawing Area. Use this window to add text to the Page Interface Editor window's Drawing Area.
	Working Grid	Displays the **Define Work Grid** window, define the size (the distance between grid lines) of the working grid. Items in the Drawing Areas of the PLUS+1 GUIDE, **Page Interface Editor, Module Viewer** windows snap to the working grid. This is an invisible grid.
	Close	Closes the **Page Interface Editor** window.

User Interface

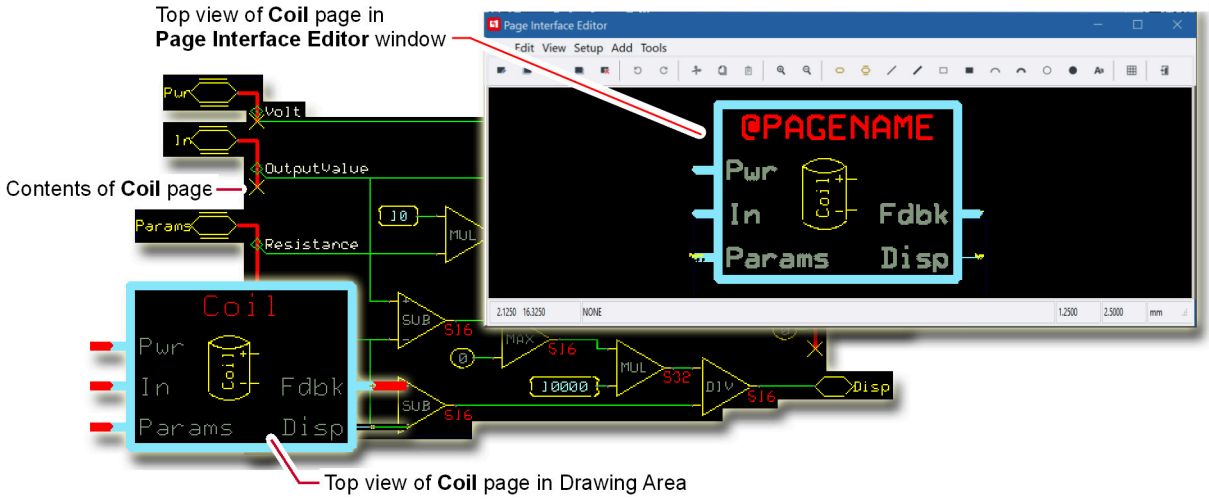
About Pages, Page Top views, and the Page Interface Editor Window



Hardware Templates provided by Danfoss contain pages, such as the **Inputs, Outputs,** and **Application** page. Some of these pages contain inner pages, such as the **MFIn** and **MFOut** pages. These pages logically organize the Hardware Template.

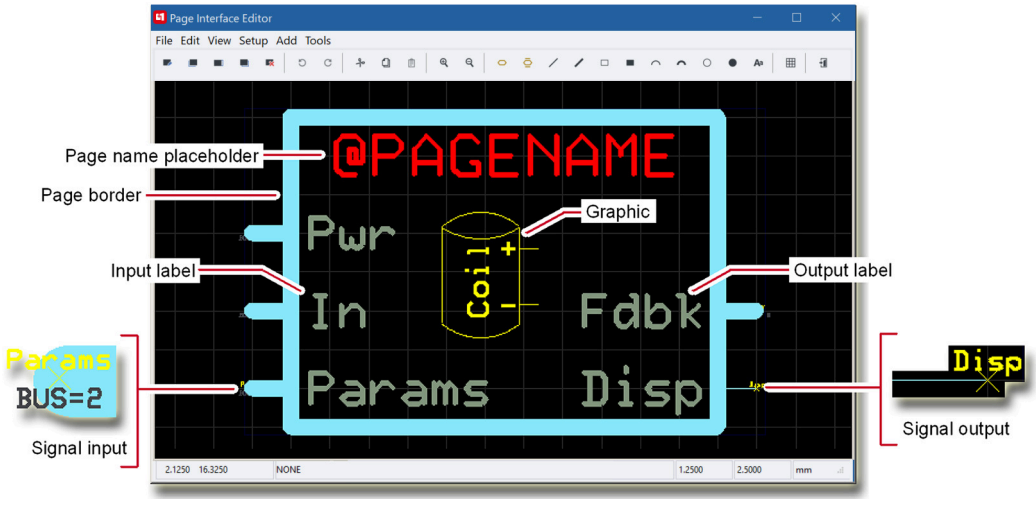
You can create your own pages to:

- create more space in which to lay out components.
- logically organize application functions.
- reduce clutter and confusion.
- create "packages" for use elsewhere within your application or export for use in other applications.



Editing the top view of a page is a basic step in making a page. The top view of a page is the exterior or shell for the page's contents; it is what you see before you enter the page. You edit the top view of a page in the Page Interface Editor window's Drawing Area.

User Interface



Use the tools in the Page Interface Editor window to:

- Draw page borders.
- Position the **@PAGENAME** placeholder for the page name.
- Add signal inputs and outputs.
- Label inputs and outputs.
- Draw graphics.

Leave the **Layer** value of the **@PAGENAME** placeholder set to the Layer value of **PageName**. If you change this value, the page name may disappear, you may have compile problems and problems with the PLUS+1 program's advanced features.

Display the Page Interface Editor window to edit the top view of:

- A new page added to the Drawing Area with the **Add** menu's **Page** command.

  For more information, see *How to Add Page with the Page Command* on page 114.
- A **Basic Page** dragged from the **Components** tab into the Drawing Area.

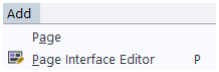
  For more information, see *How to Add a Basic Page* on page 115.
- An old page that needs changes in its size, labels, graphics, or inputs and outputs.

  For more information, see *How to Change an Old Page* on page 116.

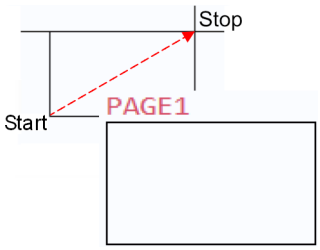
User Interface

How to Add Page with the Page Command

1. In the **Add** menu, click **Page**.

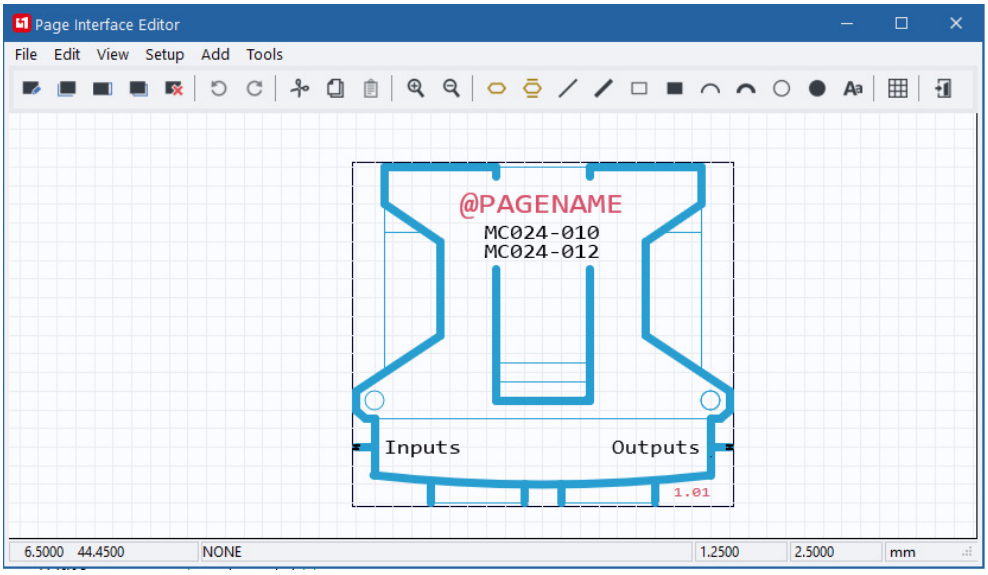


2. In the Drawing Area, add a new page by clicking and dragging from the lower left to the upper right.



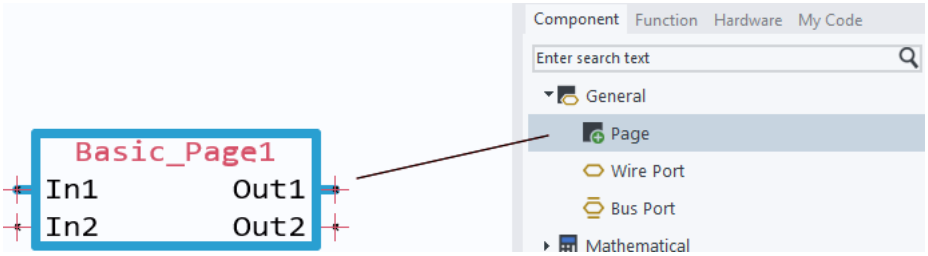
Display the Page Interface Editor window

1. In the Toolbar, click the **Enter page** button .
2. Enter the page.
3. In the toolbar, click the **Page Interface Editor** button .
4. The Page Interface Editor window displays, showing the top view of the new page in its Drawing Area.



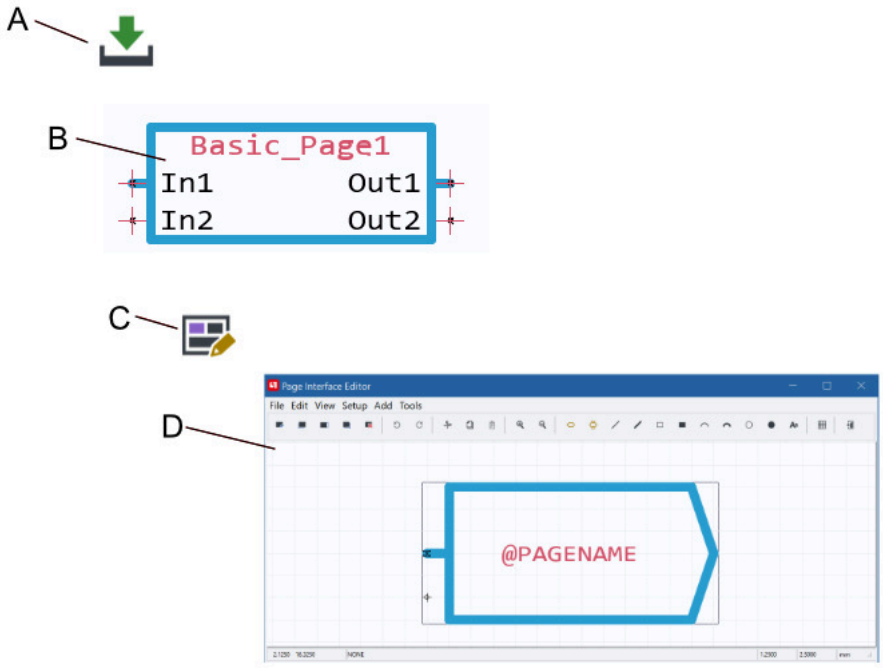
User Interface

How to Add a Basic Page



Drag a **Basic Page** from the **Components** tab into the Drawing Area.

Display the Page Interface Editor window



A — In the toolbar, click the **Enter page** button.

B — Enter the **Basic Page**.

C — In the toolbar, click the **Page Interface Editor** button.

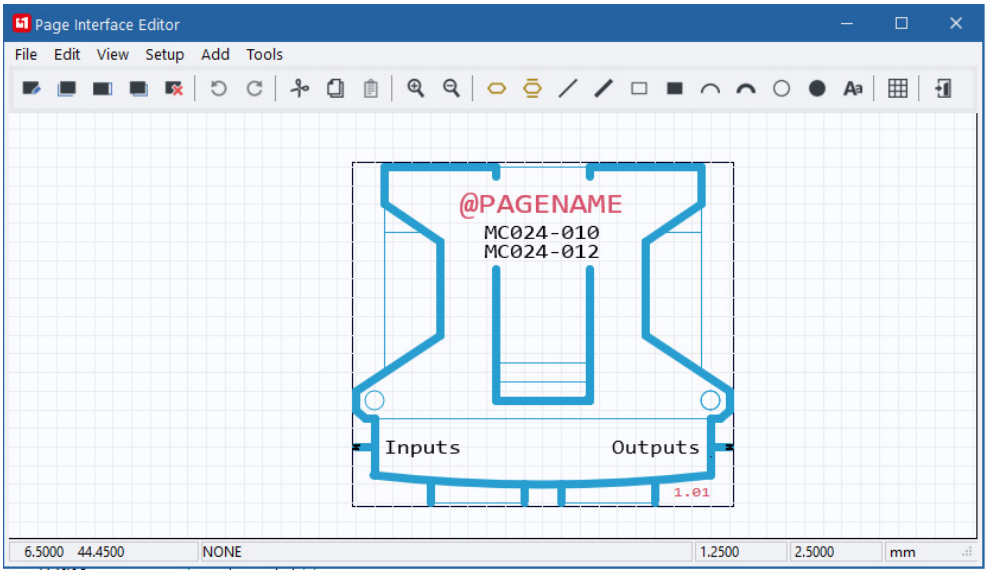
D — The Page Interface Editor window displays, showing the top view of the **Basic Page** in its Drawing Area.

User Interface

How to Change an Old Page

1. In the toolbar, click the **Enter page** button .
2. Enter the old page whose top view needs to be changed.
3. In the toolbar, click the **Page Interface Editor** button .

   The Page Interface Editor window displays, showing the old page's top view in its Drawing Area.

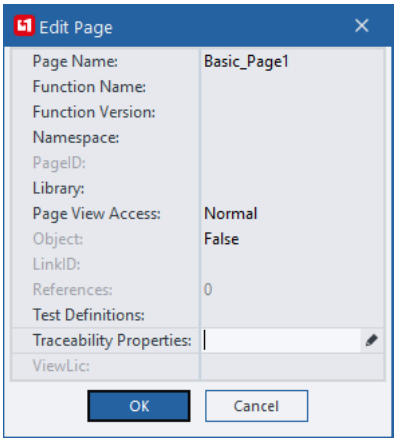


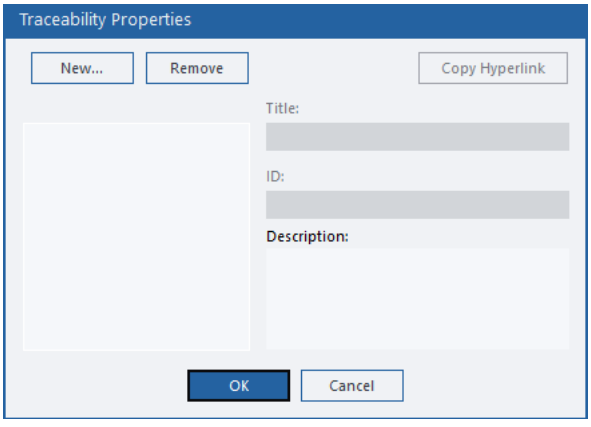
User Interface

Traceability Properties

This is a PLUS+1® GUIDE upgrade feature. A Quality Assurance License (reference PLUS+1® GUIDE add on license Quality Assurance Data Sheet, **AI0000025**) enables this feature. For more information, see PLUS +1® GUIDE Licensing, **AQ419969404812**.

Traceability Properties is accessed via **Edit Page**.





Traceability Properties window allows the user to manage traceability between requirements and implementation.

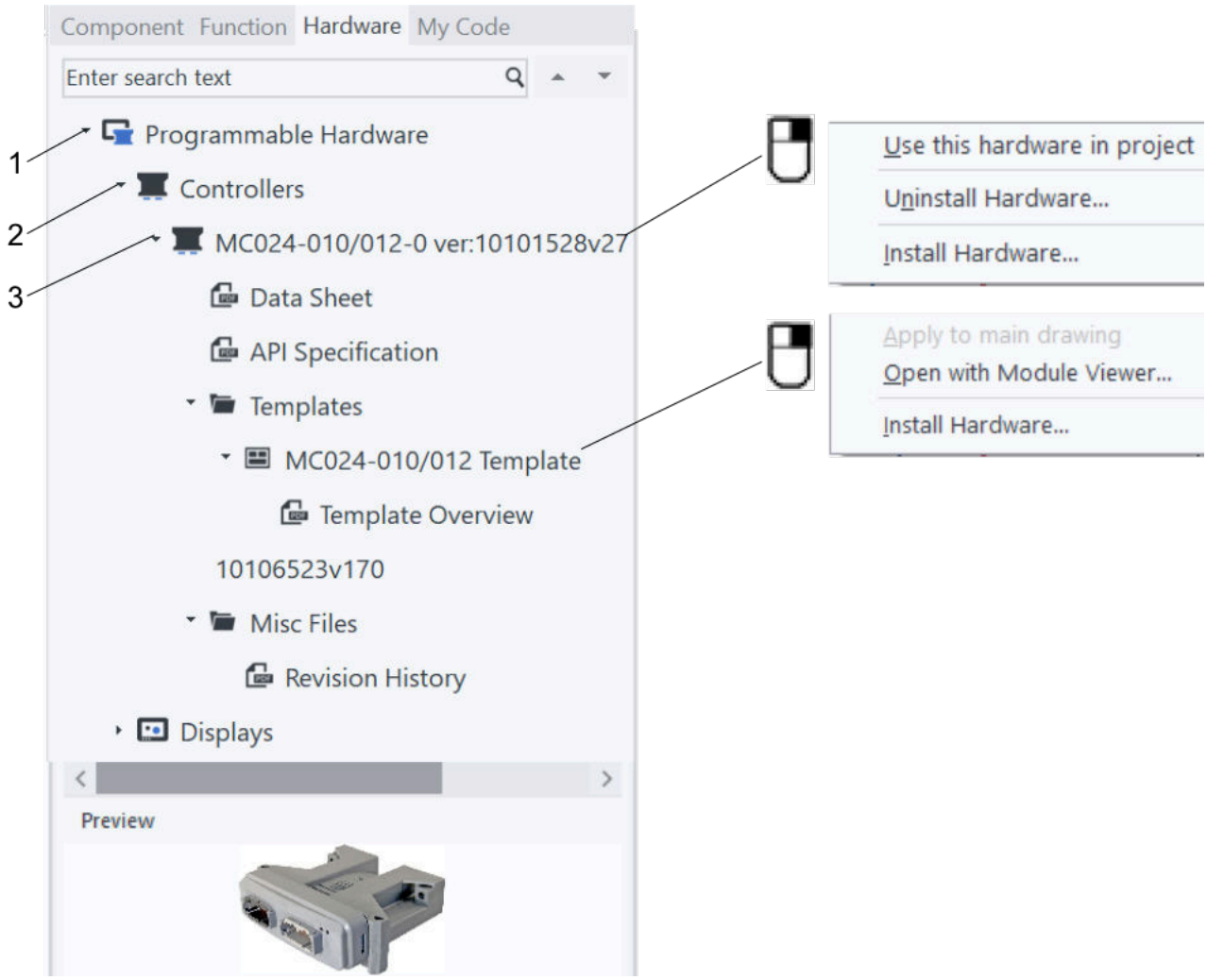
A Traceability Property includes a **Title**, an identifier (**ID**) and a **Description**. The description can also include hyperlink to external documents (requirement descriptions).

Copy Hyperlink copies a hyperlink to the current **ID** to the clipboard. This can then be pasted in an external document (requirement description) to be able to jump from requirement to the implementation.

User Interface

Tabs

Hardware



1. Hardware Category
2. Hardware Type
3. Hardware Description

***Hardware** Tab Description*

Item	Description
Search	Search for part numbers, descriptions or keywords in the **Hardware** tab tree. Jump to the previous or next search hit in the hardware structure with the buttons .
Hardware Category	These folders organize hardware by category, such as **Programmable Hardware** and **IO Modules**.
Hardware Type	These folders organize hardware by type within hardware categories.
Hardware Description	Links to the resources that the PLUS+1® GUIDE software needs to create and compile an application for a specific PLUS+1® hardware model. To install a Hardware Description, right-click the Description to display a pop-up menu with the `Use this Hardware in Project` command. Documentation for the hardware typically includes a **Data Sheet**, an **API** (Application Programming Interface) **Specification**, and a **Template Overview**.
Data Sheet	Click to view the product specification for the hardware model. This document provides a general description of the hardware model that typically includes its features, dimensions, power requirements, and pin assignments.

User Interface

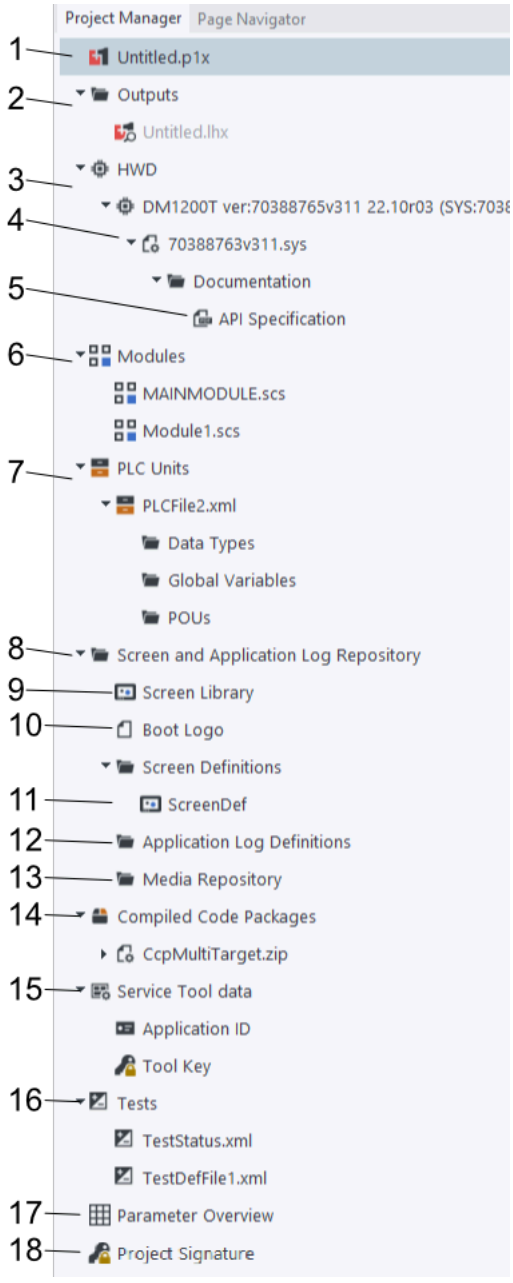
Hardware *Tab Description (continued)*

Item	Description
API Specification	Click to view the application interface (API) description for the hardware model. Refer to this document when configuring hardware inputs and outputs.
Hardware Template	Each Hardware Description is associated with a **Hardware Template**. A Hardware Template simplifies programming by having inputs and outputs that are configured appropriately for the hardware. Right-click the Hardware Template to select `Open with Module Viewer` or `Install hardware`.
Template Overview	Click to view a document that provides basic information about using the template.
Preview pane	Previews the selected hardware model. Use the **Options** window settings to enable this preview, see *Preview Settings - concept topic*.

User Interface

Project Manager

This tab lists software and hardware descriptions used in the project. The Hardware Descriptions are specific to PLUS+1® hardware models. A Hardware Description has the resources needed by the PLUS+1® GUIDE software to create and compile an application for a specific model of PLUS+1® hardware.



User Interface

Project Manager tab

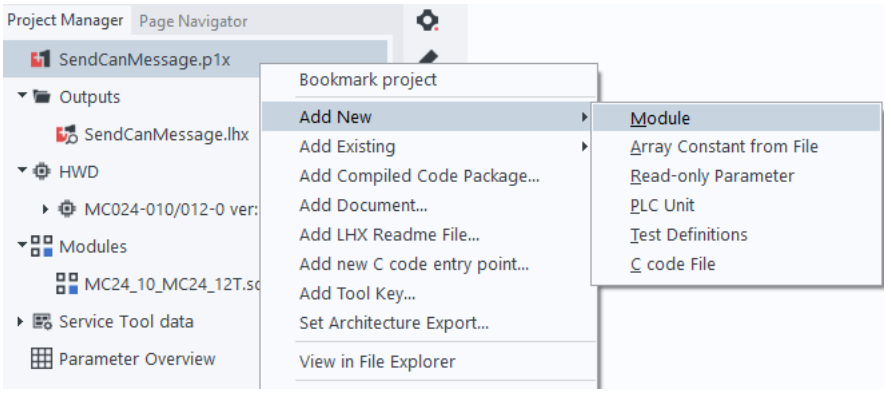
1. Project Name	The project name. A project can consist of one main module and several subsidiary modules. Right-click the **Project Name** to display a pop-up menu with the commands described more in the following table. Use the **Inspector** tab to edit the following project related settings: • Default string length, all strings in the project that do not have an explicit length defined will get the length defined by this setting. • ModuleConnectionLog, if True, all module connections will be added as checkpoints to the diagnostic data. • PouConnectionLog, if True, all variables defined in the VAR_INPUT and VAR_OUTPUT sections of POUs will be added as checkpoints to the diagnostic data.
2. Outputs	The list of files generated from the project on compile. There will always be a LHX file (Service Tool writes one to the target hardware units). Example of other compile output files: • Automatically generated P1D files (Intended for use by the application developer as a helpful diagnostic tool, not suitable for Service Technicians in the field.) • Architecture report XML files. These files contain the page structure of the project and their traceability properties, which can be used during reviews. After you have compiled your application, double-click this file to download it to the hardware. The PLUS+1® Service Tool program will open, with this file ready to download.
3. HWD, Hardware Description	The Hardware Description file contains the system file and some additional documentation files, templates, preview images, skin images, icons, etc.
4. System File (kernel)	The operating system used with the hardware. This file installs with the Hardware Description and cannot be changed.
5. API Specification	Right-click to open the **API Specification** (Application Programming Interface). This PDF document describes capabilities of the hardware, including its pin configurations, supported components, and CAN implementation.
6. Modules	The graphical code modules included in the project. The first one listed is always the mandatory main module that directly or indirectly calls all the other optional modules. Double-click to show a module in the **Drawing Area**. Right-click to add new or existing module.
7. PLC Units	PLC units in the project.
8. Screen and Application Log Repository	Vector-based screen editor/application log, media repository and boot logo. This branch and its sub-branches are hardware dependent.
9. Screen Library	Use the context menu to import or export screen library assets.
10. Boot Logo	The Boot Logo is shown on the display during start-up. It is only displayed in the structure if a boot logo image is added to the project.
11. ScreenDef	Screen definitions in the project.
12. Application Log Definitions	Application log definitions in the project.
13. Media repo	Media files in the project.
14. Compiled Code Packages	Compiled code packages in the project.
15. Service Tool data	This information is visible in the Service Tool after the corresponding LHX file has been written to a target hardware unit. Use the **Inspector** tab to edit the Application ID. The Application ID consists of four parts: • Description, by default the same as the project name but can be modified. • System ID, empty by default. The idea is that all GUIDE programmable ECUs on the same CAN bus, can be given the same System ID. This System ID can be used by the Service Tool (including Mobile Service Tool) to automatically match and find the correct diagnostic applications for a specific system. • Type, empty by default. The type field is used to make sure that the correct ECU is updated when writing a new LHX file to a target hardware unit. If the type field in the LHX and the target hardware unit do not match, the Service Tool will warn the user about this. When you update an application, the old and new Types shall match, otherwise a warning is displayed. • Version, empty by default. This field allows the GUIDE programmer to keep track of versions of their software. There is no specific version number format enforced by the GUIDE tool, the recommendation is to use an established version system, for example semantic versions.
16. Tests	Consists of the current Test *Result* XML file (`TestStatus.xml`) and one or several Test *Definition* XML files that define what to test. Double click the Test Result XML file to open the Test Tool user interface

User Interface

Project Manager tab (continued)

17. Parameter Overview	Double click to display the Parameter Overview dialog. This dialog can be used to view and modify all NV and Diagnostic signals in the application.
18. Project Signature	If you have received a GUIDE project from Danfoss, it might contain a Project Signature file (*.p1sig). This file provides a license for all licensed pages that existed in the application when it was created by Danfoss. This file cannot be modified by an end user. • New licensed pages may be added to the application by the end user, but those will not be covered by the Signature file. An end user would then need to have an explicit license for those added pages. • Licensed pages covered by the Signature file may be deleted. The Signature file will still be valid for any remaining licensed pages. • Licensed pages covered by the Signature file may be moved to other locations (page paths), but then they will no longer be covered by the Signature file. An end user would then need to have an explicit license for those moved pages. • Changing the Hardware Part Number will make the signature file not valid, but updating the same Hardware Part Number to a later version is supported.

Application Context Menu



Application Context Menu

Add New Module	Click to add a new module to the application.
Add New Array Constant from File	Click to display the **Select File** window, add an array to the application.
Add New Read-only Parameter	Click to start the process of adding a new read-only parameter file to the project. A read-only parameter file contains application values that you download to the hardware.
Add New PLC Unit	Click to add a new PLC Unit to the application.
Add New Test Definition	Click to add a new Test Definition to the application.
Add New C code File	Click to add a new C code File to the application
Add Existing	Click to add existing Module, Array Constant from File, Read-only Parameter, PLC Unit, Test Definitions or C code File.
Add Compiled Code Package	Click to display the **Select Compiled Code Package File** window, add a module containing compiled C code to your application.
Add Document	Click to display the **Select File** window, add a document—such as a Microsoft Word or PDF document—to the project folder. Click the document name in the **Project Manager** tab to open the document.
Add LHX Readme File	Click to display the **Select File** window, attach a rich-text format `rtf` file to a `lhx` (download) file. The PLUS+1® Service Tool program displays the text in the `rtf` file before the `lhx` file downloads. The `rtf` file can only contain text.
Add new C code entry point	Click to add a new C code entry point
Add Tool Key	Click to display the **Add Application Tool Key** window, restrict the PLUS+1® Service Tool program's access to application values in the hardware.
Set Architecture Export	Click to enable an Architecture report XML file being generated on compile. The report file will be added in the Project Manager under Outputs.

User Interface

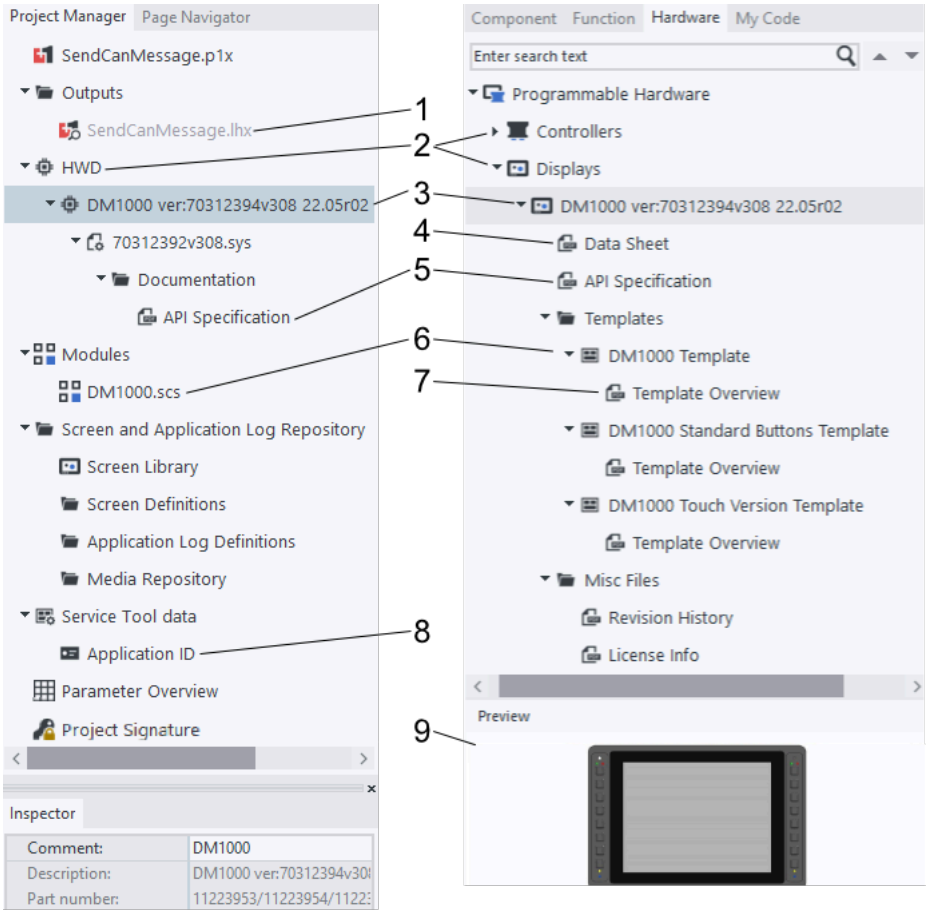
Application Context Menu (continued)

Add Boot Logo	Click to add an image that will be displayed while the display unit boots up. This menu option is only available for some display HWDs.
View in File Explorer	Click to open a Windows File Explorer Window with the corresponding file being pre-selected.

User Interface

About the Project Manager and Hardware tabs

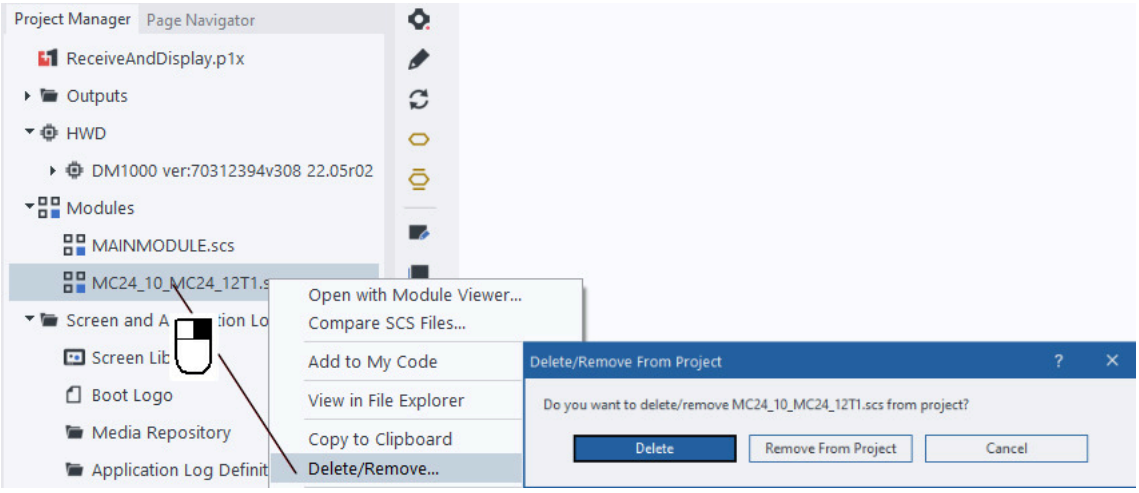
The **Hardware** tab contains the hardware-related resources needed by the PLUS+1® GUIDE software to create and compile an application for PLUS+1® hardware. Some parts can also be accessed on the Project Manager tab.



Item	Description
1. Download File	After compilation, you download a file to your hardware. This is a limited hex (`*.lhx`) format file.
2. Hardware Description	Has the resources needed by the PLUS+1® GUIDE software to create and compile an application for a specific model of PLUS+1® hardware. In the **Hardware** tab they are sorted in category. You start a Description installation by right-clicking the desired Description in the **Hardware** tab.
3.Hardware	Hardware shown in both tabs.
4. Data Sheet	Provides an overview of the hardware.
5. API Specification	Application Programming Interface (API) — describes the selected hardware's capabilities, including its pin configurations, supported components, and CAN implementation.
6. Hardware Template	The starting point for creating an application for a specific piece of hardware. It has inputs and outputs that are configured appropriately for the selected hardware. You start a Hardware Template installation by right-clicking the desired Template in the **Hardware** tab.
7. Template Overview	Describes how to use the Template.
8. Application ID	Use the Inspector tab to edit the Description, Type, Version and System ID of your application. When you update an application, the old and new Types should match. Otherwise you will get a warning before downloading the updated application.
9. Preview	Preview of the hardware.

User Interface

How to Remove Items from the Project Manager Tab

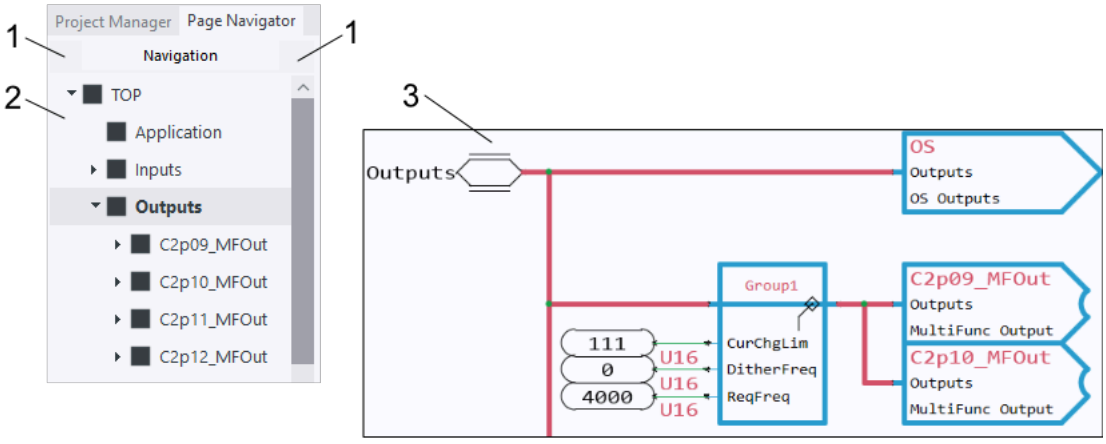


To remove an item in the **Project Manager** tab:

1. Right click an item.
2. Select **Delete/Remove...**
   - Click **Delete** to remove the item from the **Project Manager** tab and also delete its file from the project folder.
   - Click **Remove From Project** to remove the item from the **Project Manager** tab but keep its file in the project folder.

User Interface

Page Navigator



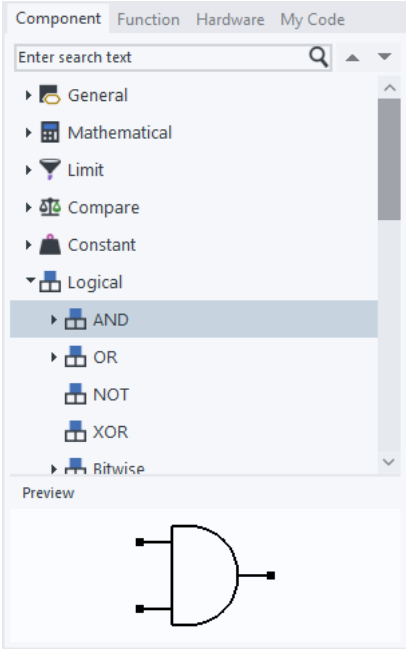
Use this tab to go directly to pages in your application.

Page Navigator Tab description

Item	Description	
Navigation buttons	Click to display previous navigation selections in the **Drawing Area**.	
Navigation pane	Displays a tree view of all the pages within your application. Click a page name in the tree to display the page in the **Drawing Area**.	
	Text color indicates the Page View Access:	
	Black	= Normal
	Green	= Force Enabled
	Maroon	= Read Only
	Gray	= Disabled

User Interface

Component



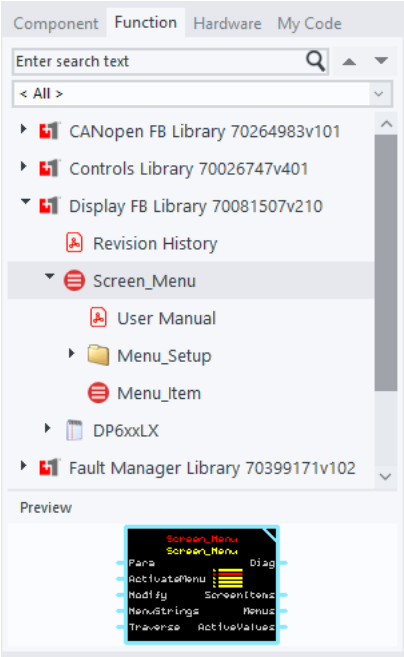
Component tab displays a tree view of components that you can drag to the Drawing Area for use in your application. Components are the basic building blocks in an application.

Component Tab Description

Item	Description
Component	Lists the components, by type, that you can use in your application. The hardware for which you are creating an application determines the components that you can use in your application. Drag the component that you want to use from here into the Drawing Area.
Preview pane	Shows a preview of the component that you have clicked. Use to show or hide this preview. See *Preview Settings - concept topic*.

User Interface

Function



Function tab displays a tree view of functions that you can drag into the Drawing Area for use in your application.

A function is a block of components, created by Danfoss, and designed to perform specific tasks such as:

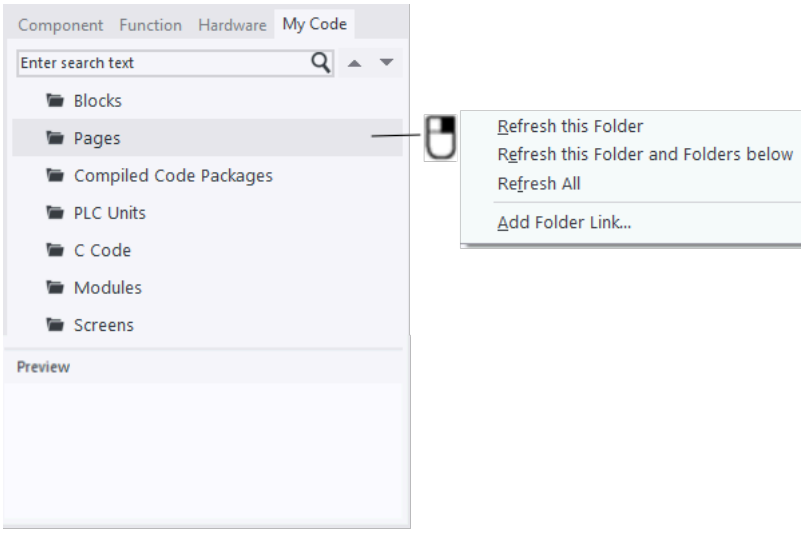
- Ackermann steering.
- Frequency-to-speed conversion.

Function Tab Description

Item	Description
Function Block Manual	Click to view the Basic Function Block Library manual.
Functions	Lists the functions, by type, that you can use in your application. Drag the function that you need from here into the Drawing Area. When new function libraries become available, use the **Install Library** command in the **Setup** menu to add them to this tab.
Preview panel	Shows a preview of the selected function. Use to show or hide this preview. See *Preview Settings - concept topic*.

User Interface

My Code



Use **My code** tab to quickly select the graphical and textual code that you have created and use frequently in your projects. This code is available for all projects.

- Several sub-folders folders store the code in this tab. The typical path to these folders is: `C:\Documents and Settings\User Name\My Documents\Danfoss\PLUS1\GUIDE\MyBlocks`.
- Use the `Add Folder Link` command to create custom folders for blocks and pages.

My Code Tab Description

Item	Description
Blocks	Lists the blocks that you have exported to the **Blocks** folder using the command: **File→Block→Export Block...** Click and drag the block that you want to use into the Drawing Area.
Pages	Lists the pages that you have exported to the **Pages** folder using the command: **File→Page→Export Page...** Click and drag the page that you want to use into the Drawing Area. The name of the page remains when it is dragged into to the application.
Compiled Code Packages	Lists legacy code packages of CCP code.
PLC Units	Lists IEC61131 code files.
C Code	Lists your C code files.
Modules	Lists your GUIDE graphical code module files.
Preview	Provides a preview of the block or page that you click.

Create a Preview Pane

To create a **Preview** pane image, perform the following procedure:

User Interface

1. Create a Windows Metafile (*.WMF) image to represent the code.

   Keep the image smaller than 150 x 150 pixels.
2. Save the WMF file in the *same folder* and with the *same file name* (except for the extension) that you used for the code file.

Create an Icon

To create an **icon** that displays next to the code file in the tree, perform the following procedure:

1. Create an icon (*.ICO) that represents the code.

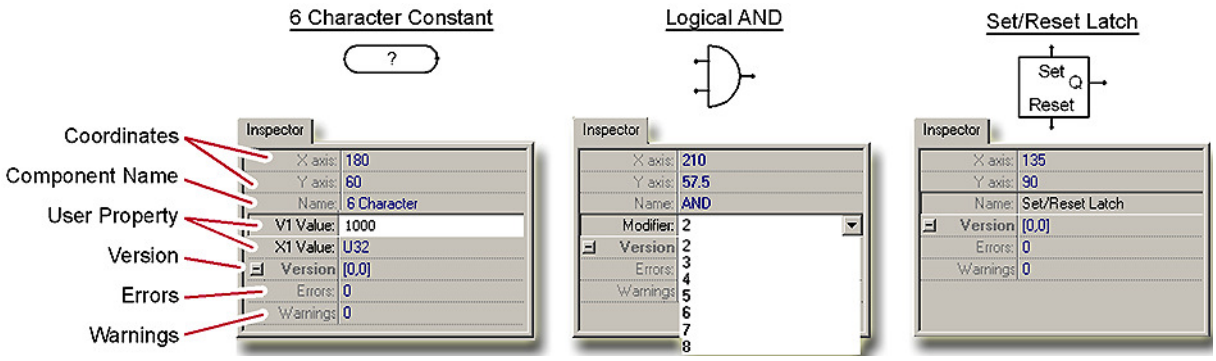
   Keep the icon smaller than 16 x 16 pixels.
2. Save the ICO file in the *same folder* and with the *same name* (except for the file extension) that you used for the code file.

For example, the *Memory.WMF* and the *Memory.ICO* files provide the preview and the icon for the *Memory.SCS* page.

User Interface

Inspector

Use this tab to view and change the properties of the items that you select with the **Query/Change** tool. The type of item that you select determines what properties you can change.

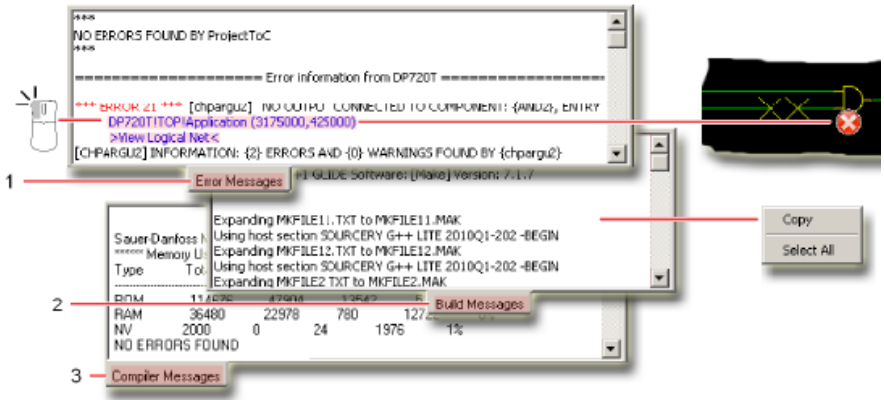


Inspector Tab Description

Item	Description
Coordinates	The x-axis and y-axis coordinates on the page where you placed the item.
Name	The name of the item.
User Property	A value that you enter to change item properties. The properties that you can change depend on the item selected. The preceding figure shows three examples of properties: • The Inspector tab for the **6-Character** constant component displays **V1** and **X1** values that set the component's value and its data type. • The Inspector tab for the **AND** component displays a **Modifier** value that sets the number of inputs on this component. • The Inspector tab for the **Set/Reset Latch** component displays no values that you can change.
Version	The release version of the item.
Errors	The error status of the item: **0** = No error. The PLUS+1® GUIDE software will compile the item without errors. **1** = Error. The PLUS+1® GUIDE software cannot compile the item. The compile process will abort with errors. You need a newer item.
Warnings	The warning status of the item: **0** = No warning. The PLUS+1® GUIDE software will compile the item without warnings. **1** = Warning. The PLUS+1® GUIDE software will compile the item, but with warnings. You should get a newer item as soon as possible.

User Interface

Compiler Messages



The **Compiler Messages** tab has:

- **Error/Warning/Hint Messages** tab
- **Build Messages** tab
- **Compiler Messages** tab.

Use these tabs to observe the compile process that produces a download file, find and fix the errors that abort the compile process.

When the compile process aborts, first check the **Error/Warning/Hint Messages** tab for errors. Next check the **Compiler Messages** and **Build Messages** tabs for errors.

Compiler Messages Tabs Description

Item	Description
Error/Warning/ Hint Messages tab	The compiler displays compile error messages in this tab. Typically, most errors produce messages in this tab. If your compile process aborts, look in this tab first. Most error messages have lines that describe the type of error and the page and coordinates where the compiler found the error. Click these lines to display the error source page in the **Drawing Area**. If the compile process aborts, click the `View Logical Net` command to open the View Logical Net window, which displays the page on which the compiler found the error. The page where the compiler found the error and the page that caused the error are not always the same. You may have to return to a higher-level page to find the source of the error. If the application uses a separate build folder, there will be a link to the build folder at the end of the messages.
Build Messages tab	The compiler displays build messages in this tab as it compiles your application into a downloadable file. This tab displays build-related errors.
Compiler Messages tab	The compiler displays the messages that it logs to the `Screen.tmp` file, located in your build folder (or project folder, if the application does not use a separate build folder). This tab displays compiler-related errors.

User Interface

Test Tool

This is a PLUS+1® GUIDE upgrade feature. A Quality Assurance License (reference PLUS+1® GUIDE add on license Quality Assurance License Data Sheet, **AI170686484256**) enables this feature.

For more information, see PLUS+1® GUIDE Licensing, **AQ419969404812**.

Use the **Test Tool** to perform tests on GUIDE code. The **Test Tool** requires no hardware and should only be used to test GUIDE code.

The first time **Test Tool** is run (generated dll-file), there may be a virus scan with an associated pop-up notification.

Creating Test Cases for a Page

The **Test Tool** creates page-level **Test Cases**.

Each **Test Case** has a **Test Case Definition Table** with columns for the values that you want to input when you execute the **Test Case** and values that expect be output as a result of the test execution.

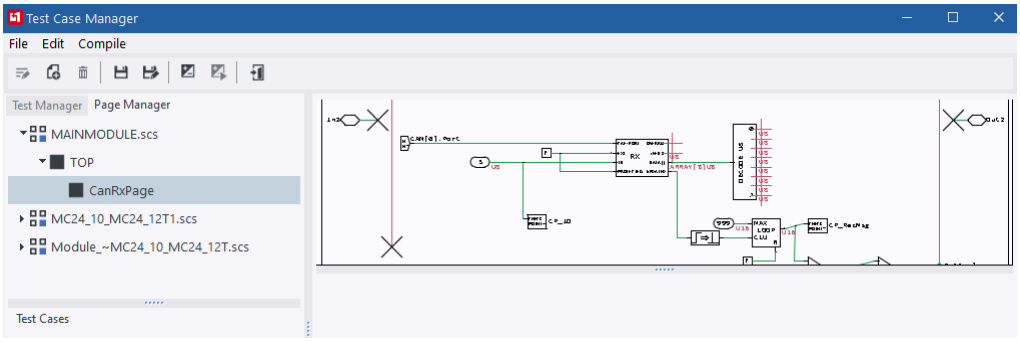
Assigning a **Test Case** to a page links the **Test Case** with the page.

Use the following components or ports to input into a page and to output values from a page:

- Hardware Input component
- Initialize Hardware Input component
- Hardware Output component
- Wire port component
- Module Single Wire components
- Set Value component
- Checkpoint components
- CAN components
- Bus port component
- Module Bus components
- Nonvolatile memory components

Creating test cases for a page

1. Click the **Tools menu** > **Test Tool**.
2. Use the **Test Case Manager** window to:



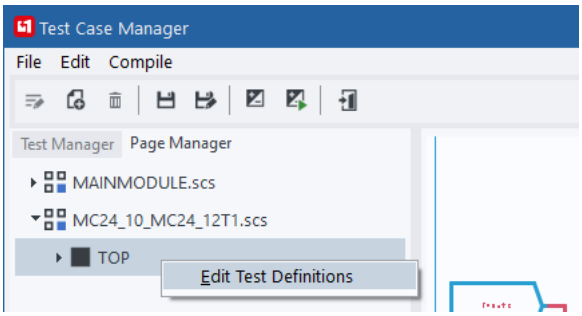
   - assign **Test Cases** to pages.
   - create a **Test Case** from scratch, without reference to an existing page.
   - generate **Test Case Definition Tables** for **Test Cases**.
   - add input values and expected output values to **Test Case Definition Tables**.
   - execute **Test Cases** and review the results.

   For more information, see *Test Case Manager Window—Menus and buttons* on page 136.
3. The **Page Manager** tab:

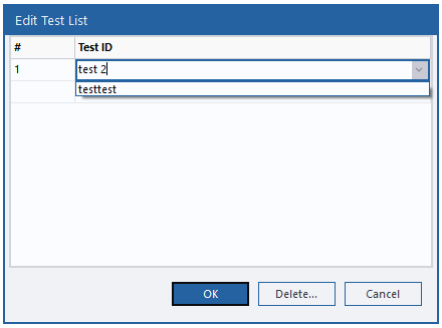
User Interface

- mirrors the functionality of the GUIDE window's **Page Navigator** tab.
- provides a way to assign **Test Cases** to pages through the **Edit Test List** window.

4. Right-click a page in the **Page Manager** tab and select **Edit Test Definitions**.

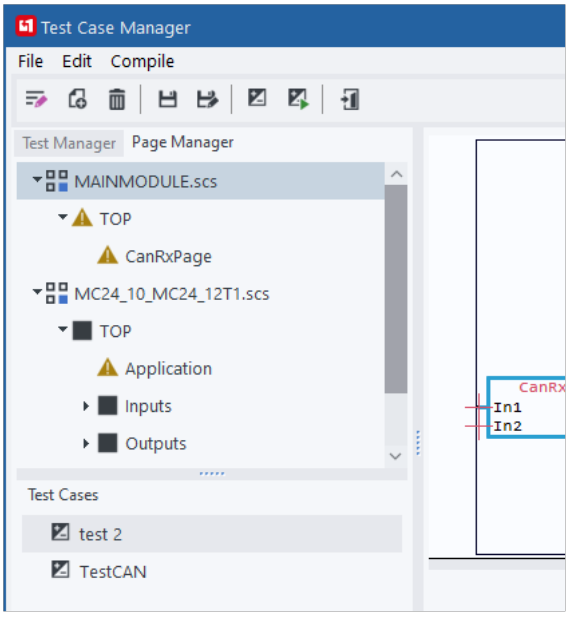


5. Use the **Edit Test List** window to:



- assign a **Test Case** to a page that you highlighted in the **Test Manager** tab.
- create a **Test Case** from scratch, without reference to an existing page.
- delete a **Test Case** assignment from a page that you highlighted in the **Test Manager** tab.

6. The **Test Cases** tab, lists the test cases that you have assigned to the selected page.



7. The following icons in the **Page Manager** tab indicate the status of each page.

⚠ — You need to generate a **Test Case Definition Table** for the Test Case assigned to the page.

User Interface

— All Test Cases assigned to the page have successfully executed.

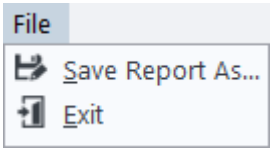
— One or more of the Test Cases assigned to the page have failed.

— The Test Case assigned to the page does not exist.

User Interface

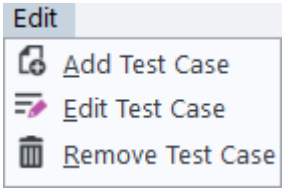
Test Case Manager Window—Menus and buttons

File menu



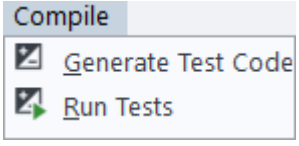
Save Report As	Use the **Save Report As** window to save the report files available in the **Test Manager** tab view. These files report test results.
Exit	Exits the **Test Case Manager** window.

Edit menu



Add Test Case	Use the **Edit Test List** window to: • Assign a Test Case to a page that you highlighted in the **Test Manager** tab. • Delete a Test Case assignment from a page that you highlighted in the **Test Manager** tab.
Edit Test Case	Opens a Test Case Definition Table for editing.
Remove Test Case	Deletes a Test Case assignment from a page that you highlighted in the **Test Manager** tab.

Compile menu



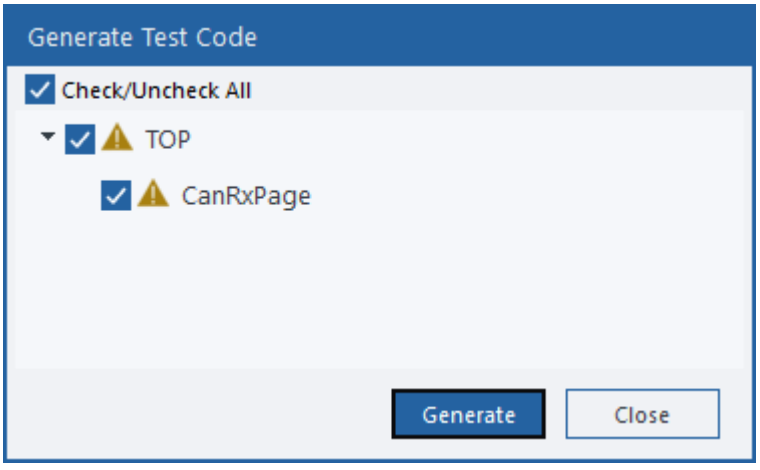
Generate Test Code	Use the **Generate Test Code** window to select the pages for which you want to create Test Case Definition Tables. You must assign a Test Case to a page to be able to generate a Test Case Definition Table for the page.
Run Tests	Runs tests for all pages with Test Case Definition Tables.

User Interface

Generating a Test Case Definition Table for a Page

- To run a test, create a Test Case Definition Table for each Test Case that you have assigned to an existing page.
- The Test Tool generates Test Case Definition Tables based on the inputs and outputs of the page under test.
- The Test Case Definition Table generated by the Test Tool has columns and rows for you to enter test input values and your expected output values.

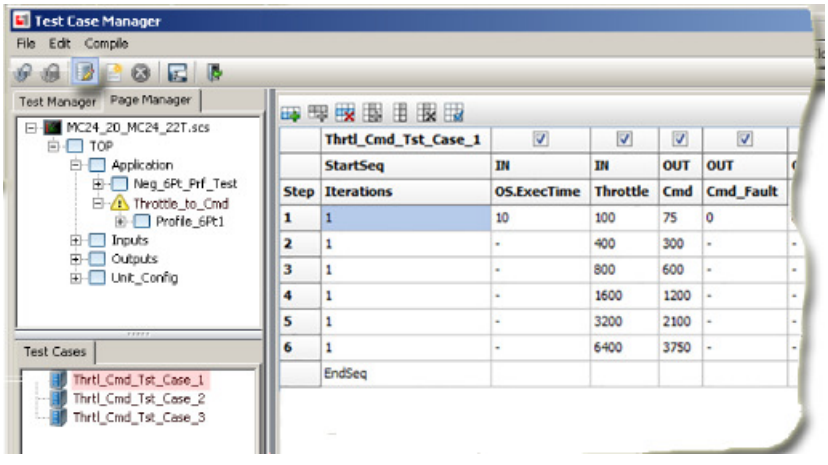
1. Click **Generate Test Code**.



2. The **Generate Test Code** window mirrors the page structure shown in the **Page Manager** tab of the main **PLUS+1 GUIDE** window.

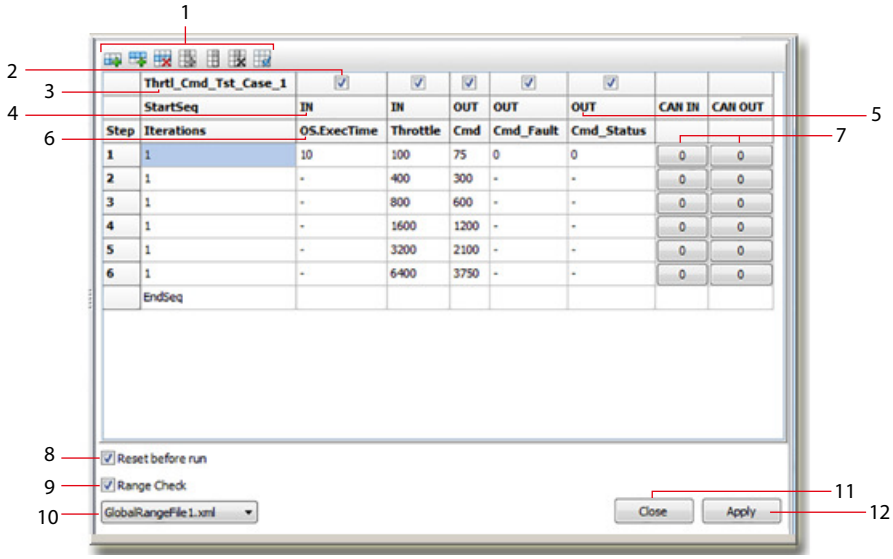
   Click the page or pages for which you want to generate Test Case Definition Tables.

   (You must first assign a Test Case to a page to be able to generate a Test Case Definition Table for the page.)
3. Click the **Generate** button in the **Generate Test Code** window to generate Test Case Definition Tables for the selected pages.
4. To view a **Test Case Definition Table** for a Test Case:
   a. Click a Test Case in the **Test Case** tab.
   b. Click the **Edit Test Case** button.



User Interface

Test Case Definition table



Use the Test Case Definition Table to enter:

- the Values that you want to apply in a test.
- the Values that you expect to output as a result of your input values.

Test Case Definition table description

Item	Button/column/checkbox	Description
1.	Buttons	The **Page Manager** tab view makes these buttons available: • **Add Row** — Click to add a new bottom row to the Test Case Definition Table. • **Insert Row**— Click to add a new row above selected row in the Test Case Definition Table. • **Delete Row** — Click to delete a selected row from the Test Case Definition Table. • **Show/Hide Excluded Signals** — Click to switch between a view with columns for: — Both included and excluded signals. You see both checked and unchecked columns. — Only included signals. You see only checked columns.
2.	**Include/ Exclude** checkbox	Check to include a column's signal in a test. Uncheck to exclude a column's signal from a test.
3.	**Test Case** name	The name of the Test Case.
4.	**IN** columns	**IN** identifies columns with the signal values to be input when you execute the test. Enter the sequence of values that you want to apply in the rows of the **IN** columns. Enter the number of times that you want to apply a value in the **Iterations** column.
5.	**Expected** columns	**Expected** identifies columns with the signal values that you expect to be output when you execute the test. Enter the sequence of values that you expect to be output in the rows of the Expected columns. In previous versions, this was called **'OUT'.**
6.	**OS.ExecTime** column	The execution time in milliseconds per Iteration.
7.	**CAN IN**, **CAN EXPECTED** buttons	• Click to open an **Edit CAN Messages** window for each row in the **CAN In** column. • Click to open an **Edit CAN Messages** window for each row in the **CAN Expected** column. Use these windows to enter the sequence of **CAN messages** that you: • Want to input when you execute the test. • Expect to be output when you execute the test. In previous versions, **'CAN Expected'** was called **'CAN OUT'.**
8.	**Reset before run** checkbox	Click to reset all inputs and internal states to zero at the start of the test.
9.	**Range Check** checkbox	Click to apply a range check during the test. A range check sets the upper and lower limits for one or more inputs or outputs. A test fails when an input or output exceeds an upper or lower limit.

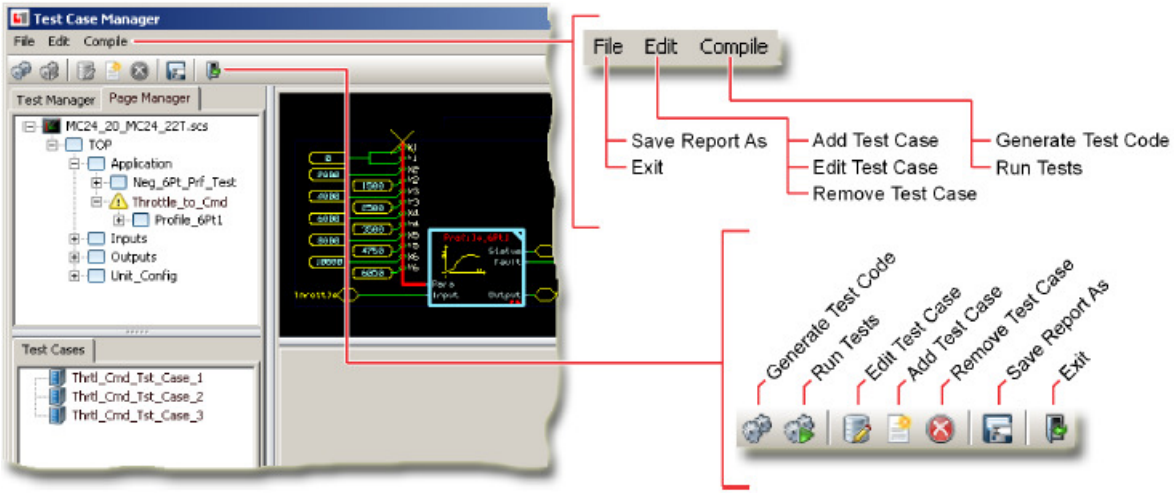
User Interface

Test Case Definition table description (continued)

Item	Button/column/checkbox	Description
10.	**Range selection** drop-down list	Selects a **Range Check** file to apply during a test. A Range Check file sets the allowable upper and lower limits for an input or output. For more information, see *About the Test Manager Tab View* on page 142.
11.	**Close** button	Click to close the Test Case Definition Table. You must first click the **Apply** button to apply any changes that you made in the Test Case Definition Table.
12.	**Apply** button	Click to apply changes made in the Test Case Definition Table but leave the Table open.

User Interface

Test Case Execution and Test Results



Legend:

Click **Run Tests** button to execute Test Cases.

Test Results window summarizes the results of all Test Cases.

Result tab shows the result of the Test Case in a table.

Graph tab shows the result of the Test Case in a graph.

Description tab enter comments and details about the Test Case.

— All Test Cases assigned to the page have successfully executed.

— One or more of the Test Cases assigned to the page have failed.

The Test Case identified in the cell passed.

The Test Case identified in the cell failed.

Click to view the results of the other Test Cases that you have assigned to the page.

User Interface

About Test Case Results

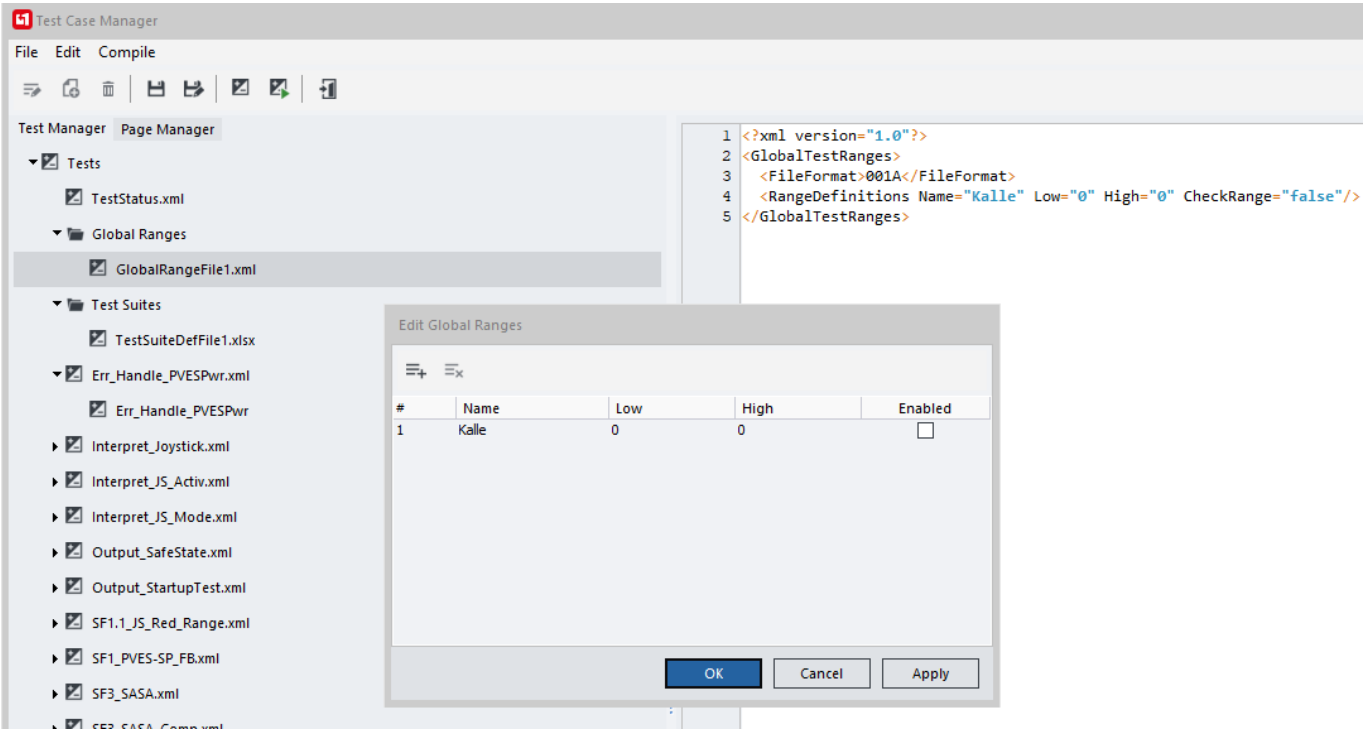
Use the **Result, Graph,** and **Description** tabs to review and comment on the results of Test Cases.

Figure details

Item		Description
1.	**Result** tab	Shows the results of a Test Case in a table format.
2.	Passed Failed	The Test Case identified in the cell passed. The Test Case identified in the cell failed.
3.	**In** columns	**In** identifies columns with the signal values that input when the Test Case executed. You entered these values before the you executed the Test Case. Values in the **Iteration** column indicate the number of times a value input when the Test Case executed.
4.	**Expected** columns	**Expected** identifies columns with the signal values that you expected to output when the Test Case executed. You entered these values before you executed the Test Case.
5.	**Actual**	Identifies columns with the **Actual** (measured) signal values that output when the Test Case executed. Red-bordered cells show where measured output values differ from **Expected** values.
6.	Passed Failed	The individual Test Case step passed; the **Actual** value equals the **Expected** value. The individual Test Case step failed; the **Actual** value does not equal the **Expected** value.
7.	**Graph** tab	Shows the Test Case result in a graph.
8.	**Show** checkbox	• Check to show a selected signal in the graph. • Uncheck to hide a selected signal in the graph.
9.	**Description** tab	Enter comments and details about the Test Case.

User Interface

About the Test Manager Tab View



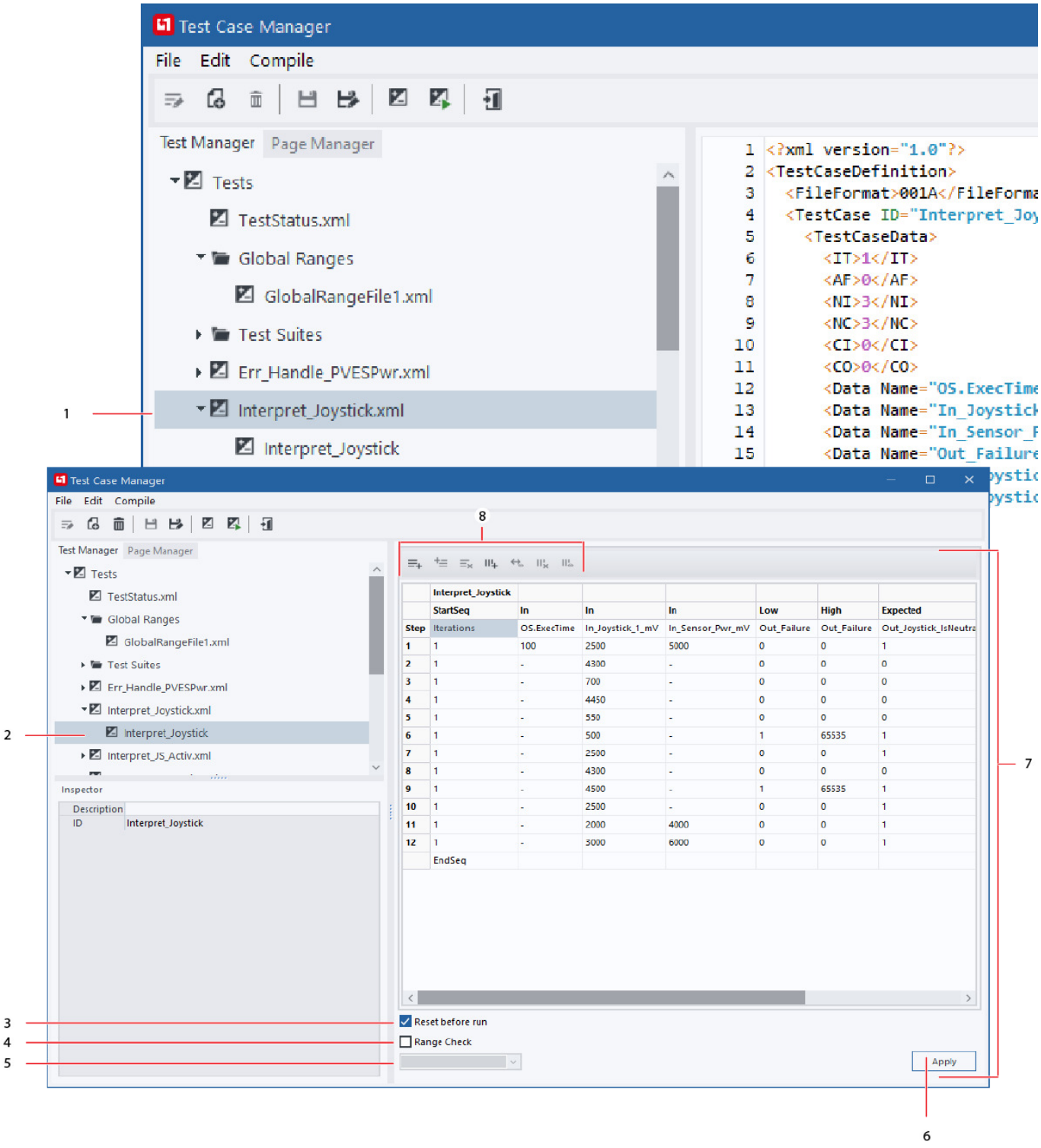
Use the **Test Manager** tab view to:

- Review and save read-only XML files that report on test status.
- Define global ranges to apply to **Test Cases**.
- View and edit **Test Cases** in the **Test Case Definition Tables**.
- Define and edit **Test Suites**.

Test Case Definition Table description

Item	Description
Test Manager tab	Mirrors the structure of the Test node in the PLUS+1® GUIDE window's **Project Manager** tab.
Test Status.xml node	Click this tab to view a read-only XML file that reports the results of the last execution of the **Test Case**. Each execution of Test Cases updates this file. Use the **File** > **Save Report as** command to save this report.
Global ranges node	Right-click to open a pop-up menu with **Global Range File** commands. • Click the **Add Global Range File** command to open the **Edit Global Ranges** window. Use the **Edit Global Ranges** window to set the upper and lower limits for one or more outputs. A test fails when an output exceeds an upper or lower limit. • Click the **Add Existing Global Range File** command to open the **Select GlobalVars File** window. Use the **Select GlobalVars File** window to select an existing global range file.
Test Suite definitions node	Right click to open a pop-up menu with **Test Suite File** commands. • Click the **Add new Text Suite definitions file** to add a new file of either xml or excel file format. • Click the **Add existing Text Suite definitions file** to add an existing test suite file.

User Interface

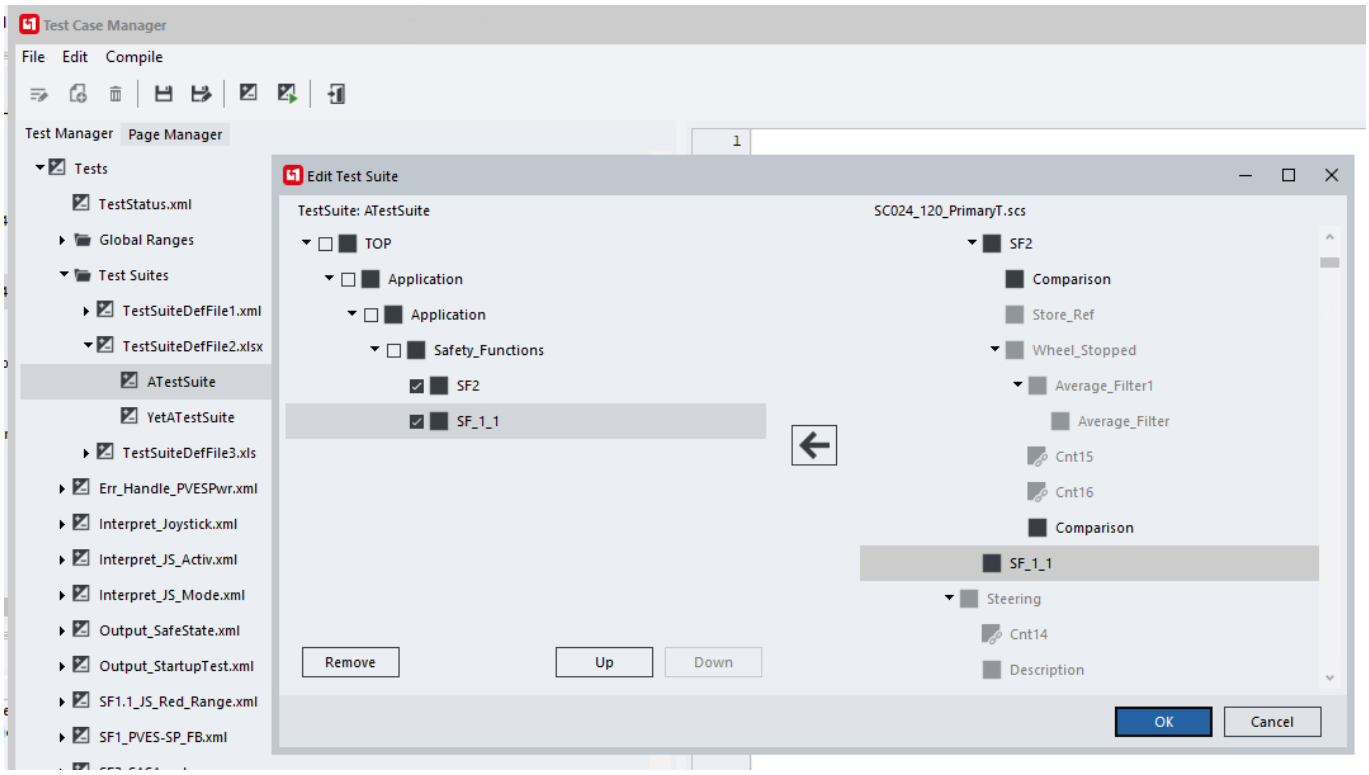


User Interface

Test Case Definition Table description

Item		Description
1	**TestCaseDefinition** node	Click the read-only XML file that reports the applied inputs and expected outputs of each **Test Case**. Go to **File** > **Save Report As** command to save this report.
2	**Test Case** file	Click a **Test Case** file under **Test Manager** tab to open the **Test Case Definition Table** for that selected **Test Case**.
3	**Reset before run** checkbox	Click **Reset before run** checkbox to reset all inputs and internal states to zero at the start of the test.
4	**Range Check** checkbox	Click **Range Check** checkbox to apply a range check during the test. A range check sets the upper and lower limits for one or more inputs or outputs. A test fails when an input or output exceeds an upper or lower limit.
5	**Range** drop-down list	Select a Range file from the drop-down list to apply during a test. A range check file sets the allowable upper and lower limits for an input or output. For more information, see *About the Test Manager Tab View* on page 142.
6	**Apply** button	Click to apply changes made in the **Test Case Definition Table** but leave the **Test Case Definition Table** open.
7	**Test Case Definition Table**	Use the **Test Case Definition Table** to create, edit and review a selected **Test Case**.
8	Buttons	The **Page Manager** tab view makes these buttons available: • **Add Row** — Click to add a new bottom row to the **Test Case Definition Table**. • **Insert Row**— Click to add a new row above selected row in the **Test Case Definition Table**. • **Delete Row** — Click to delete a selected row from the **Test Case Definition Table**. • **Show/Hide Excluded Signals** — Click to switch between a view with columns for: — Both included and excluded signals. You see both checked and unchecked columns. — Only included signals. You see only checked columns.

A **Test Suite** is a suite of pages whose test cases are to be run as an entity. Use the test suite editor to select which pages shall be included in the test suite. Please note that pages can only be selected from the active module in the page manager. (The last viewed module.)



User Interface

Test Suite Table description

Item	Description
Test Suite node	This node contains the test suites files. Choose to add or create a test suite file by right-clicking the node. Available file formats are xml, xls and xlsx.
Test Suite file	Choose to create a new test suite by right-clicking the **Test Suite** file node.
Test Suite	Right-click and select to edit (or double click) a **Test Suite**.
Left pane	**Left pane** contains pages whose test cases shall be run in the suite.
Add page button	Adds a highlighted page to the concerned **Test Suite**.
Right Pane	**Right Pane** contains the page structure of the module, the pages containing **Test cases** are highlighted.
Remove button	Removes selected page from **Test Suite**.
Up button	Moves page up in the execution order of the **Test Suite**.
Down button	Moves page down in the execution order of the **Test Suite**.
OK /**Cancel** buttons	**Cancel** closes the editor and **OK** saves the **Test Suite**.

User Interface

Debugger Tool

Toolbar menu > **Run button**

The **Step, Step Over**, and **Step Loop** buttons also start the Debugger Tool.

Use the Debugger Tool to trace the flow of data through an application. By tracing the flow of data, you can detect and correct programming errors.

The first time Debugger Tool (generated exe-file) is run, there may be a virus scan with an associated pop-up notification.

The Debugger Tool can help you find errors in programming. You should not use the results of your work with the **Debugger Tool** to assert that your application works as intended. To assert that your application works as intended, you must fully and successfully test your application following your usual test procedures.

This is especially true when using user-written C code in a GUIDE project. Keep in mind that the debugger is using a single C compiler, but the actual application will be compiled with another, or multiple other compilers.

For GUIDE code and standard PLC code, the PLUS+1® GUIDE takes care of differences between platforms and compilers, but for user-written C code this is up to the user to handle correctly.

You can strategically place **Active breakpoints** in your application and use the **Run, Step Loop**, and **Step Over** buttons to simplify tracing the flow of data.

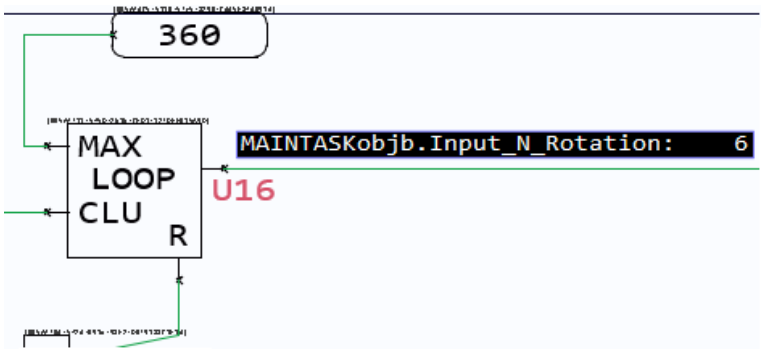
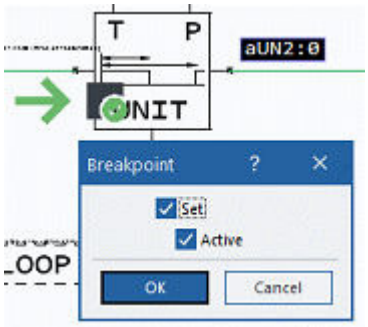
To input values to the Debugger Tool and to receive output values from the Debugger Tool, you can use the following components:

- **Hardware Input** and **Hardware Output**
- **Checkpoint**
- **Set Value** and **Set Pulse**

User Interface

Debugger Tool Elements

Debugger Tool elements description

Button/window	Description
Toolbar buttons	 Control the Debugger Tool. As needed, the Run, Step, Step Over, and Step Loop buttons first compile a Debugger Tool output before starting to control the Debugger Tool. For more information, see the table below.
Net Values	 Only displayed during a Debugger Tool session. Net Values identify the variables in your application and their current values. Use the Debugger Tool settings in the Options window to modify how the net values display or to hide the display of constant values. For more information, see *Debugger Tool Settings* on page 49.
Breakpoint window	 Query a component's insertion point to display the **Breakpoint** window. Use the Breakpoint window to create **Active** breakpoints that interrupt the execution of the Debugger toolbar's Run, Step Over, and Step Loop buttons.
Debug window	 • **Local Variables** tab — view the input and output values of the current component or PLC POU to which you have stepped the Debugger tool. • **Breakpoints** tab — use to manage and create Set, Active, and Conditionally Active breakpoints. • **Loop Input** tab — inputs values to an application while you run program loops. The GUIDE application saves these values from your Debugger Tool session for future use. • **Call Stack** tab — view the location in the application hierarchy of the current component or POU to which you have stepped the Debugger tool. • **Watches** tab — create breakpoints that trigger when the values of watched variables change. • **Loop Output** tab — records the results of program loops.

User Interface

Debugger Toolbar buttons details

Icon		Description
	Run	Runs the Debugger Tool in a non-stop cyclic loop. However, an Active breakpoint in the application interrupts this command if and when its associated component is reached. An Active Data Breakpoint also interrupts this command if and when the associated watch data value is modified. As needed, use the Pause button to manually interrupt this command or use the Stop button to completely end the Debugger Tool session.
	Stop	Stops the Debugger Tool session and closes the Debugger Tool.
	Pause	Pauses the Debugger Tool session.
	Step	Steps the Debugger Tool to the next component to be executed in the application. An Active Data Breakpoint can interrupt this command if the associated watch data value is modified by the Current component.
	Step Over	Steps the Debugger Tool to the next component (that is not located in a subpage of the current page) to be executed in the application. However, an Active breakpoint located in a subpage interrupts this command if and when its associated component is reached. An Active Data Breakpoint also interrupts this command if and when the associated watch data value is modified.
	Step Loop	Steps the Debugger Tool to the first component to be executed in the application. However, an Active breakpoint located between the current component and the first component in the execution order interrupts this command if and when its associated component is reached. An Active Data Breakpoint also interrupts this command if and when the associated watch data value is modified.

User Interface

About Breakpoints and Net Values

When the Debugger Tool stops at a component, the values that:

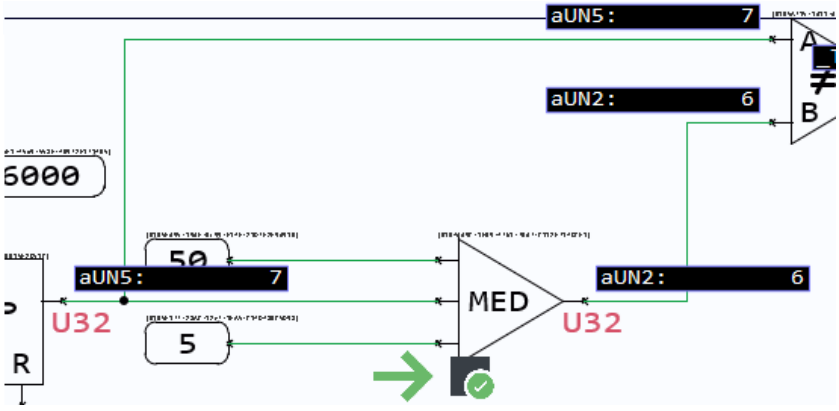
- Input to the component come from the current program loop.
- Output from the component come from the previous program loop.

When you step the Debugger Tool past the component, the values that:

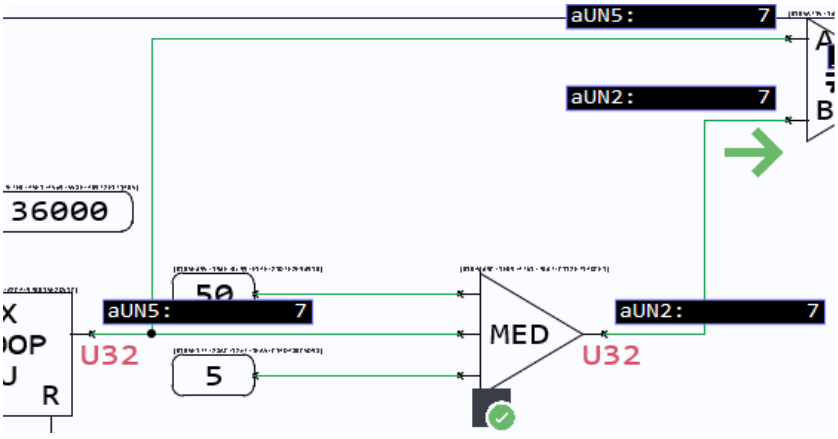
- Input and output from the component both come from the current program loop.

Legend:

1. With the Debugger Tool stepped to the **Med** component, the values that:
   - Input to the **Med** component come from the current program loop.
   - Output from the **Med** component come from the previous program loop.



2. Click the **Step** button to:
   - Execute the **Med** component.
   - Step the Debugger Tool to the **Not Equal** component (the next component in the application).



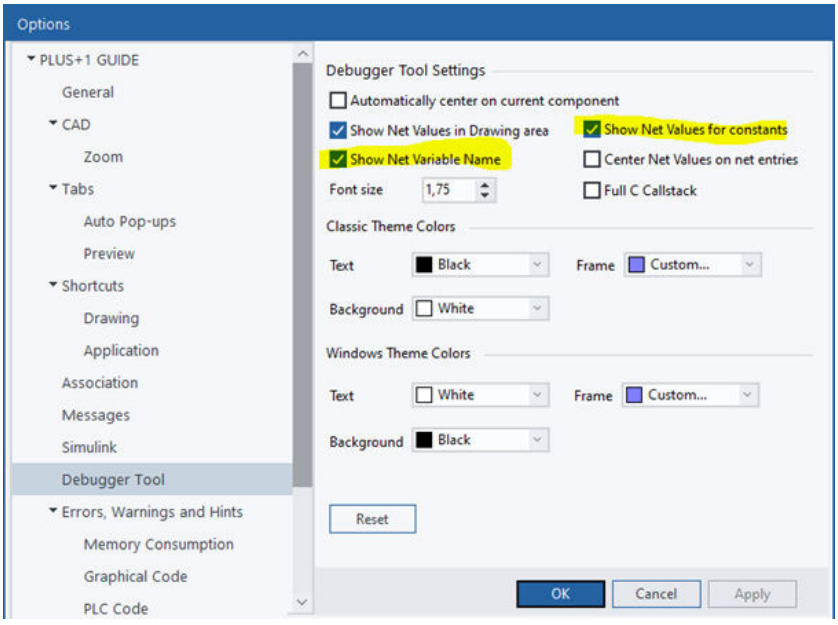
3. With the Debugger Tool stepped to the **Not Equal** component, the values that:
   - Input to the **Med** component come from the current program loop.
   - Output from the **Med** component also come from the current program loop.

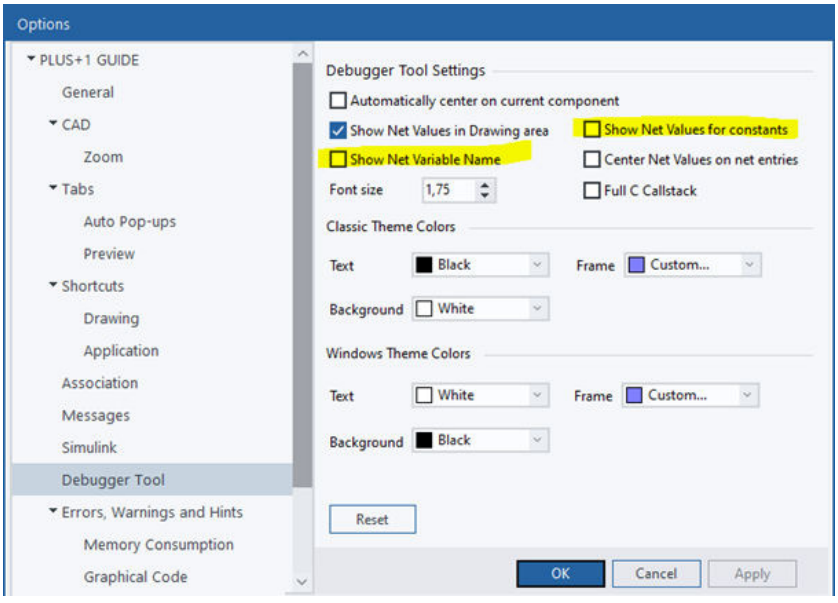
User Interface

About the Display of Net Values

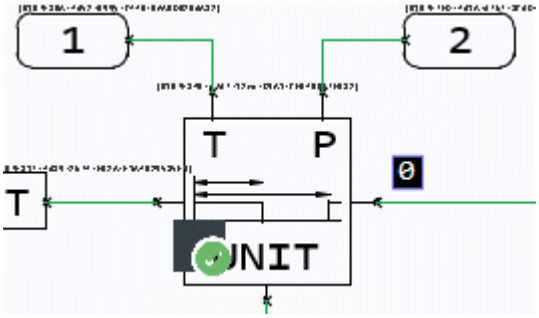
Settings > **Options** > **PLUS+1 GUIDE tree** > **Debugger Tool Settings**

Debugger Tool Settings windows





User Interface



Use **Options** > **Debugger Tool Settings** to change the display of the Debugger Tool values. For more information, see *Debugger Tool Settings* on page 49.

User Interface

About Set Breakpoints

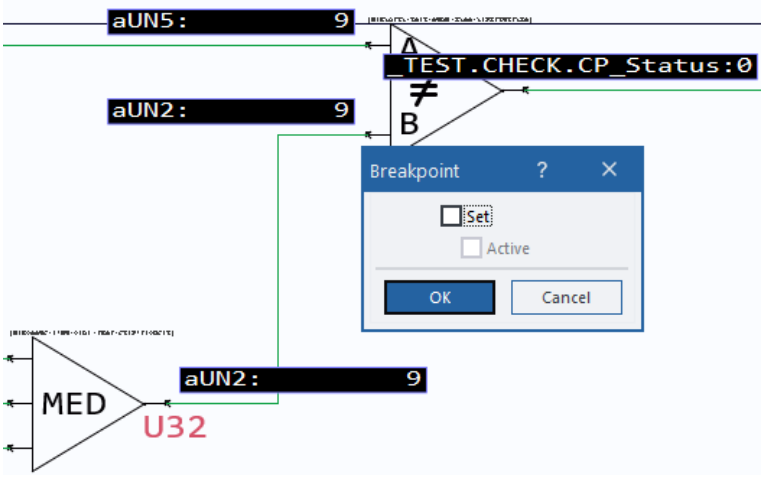
Use the **Breakpoint** window to **Set** and make **Active** breakpoints. Press **Q** (Query) to open the Breakpoint window, click-and-drag on a component. The following procedure describes how to place a Set breakpoint icon.

1. Click the **Set** checkbox in the Breakpoint window. A Set breakpoint icon appears at the component's insertion point but does not otherwise change the behavior of the Debugger Tool buttons.
2. Click **OK** button.

A **Set** breakpoint icon marks a breakpoint with an identifying icon. Otherwise, a component with a Set breakpoint acts like an ordinary component.

In some cases there will not be any C code generated for a specific element in the user code (for example due to optimization). This applies to "Show Screen" for GUIDE code, and Input and Output components in FBD/LD code, but can apply in other situations as well. In these cases, a breakpoint added on such components will be removed automatically when debugging resumes.

Breakpoints window



About Active Breakpoints

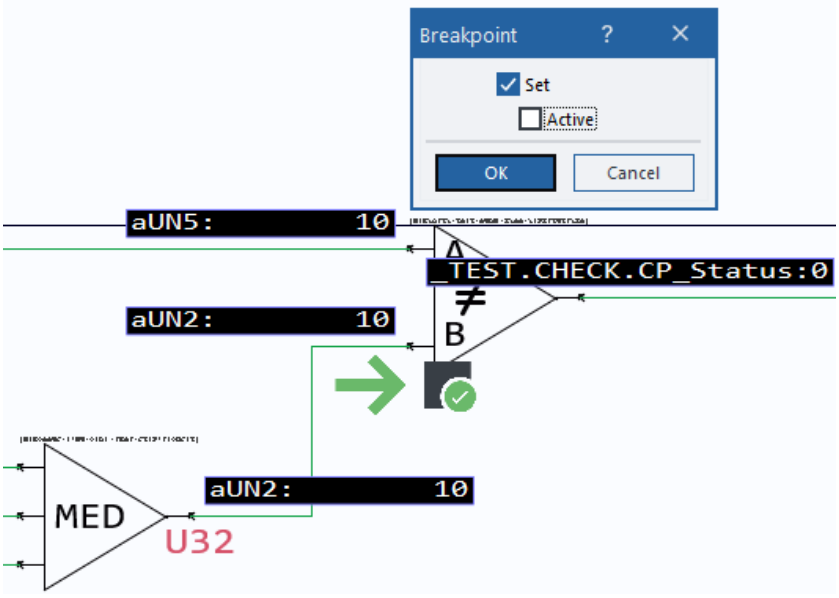
An **Active** breakpoint creates an Active breakpoint with an identifying icon. A component with an Active breakpoint interrupts the execution of the Debugger toolbar's Run, Step Over, and Step Loop buttons.

The following procedure describes how to place an Active breakpoint icon 

User Interface

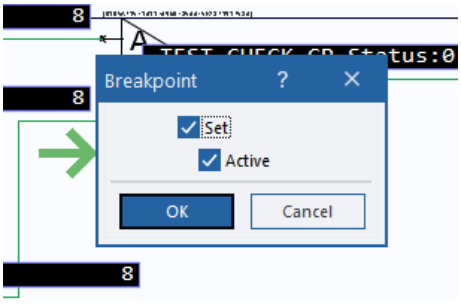
1. Click the **Set** and **Active** checkboxes in the Breakpoint window. An Active breakpoint icon appears at the component's insertion point and interrupts the execution of the Debugger toolbar's Run, Step Over, and Step Loop buttons.

   Set breakpoint figure

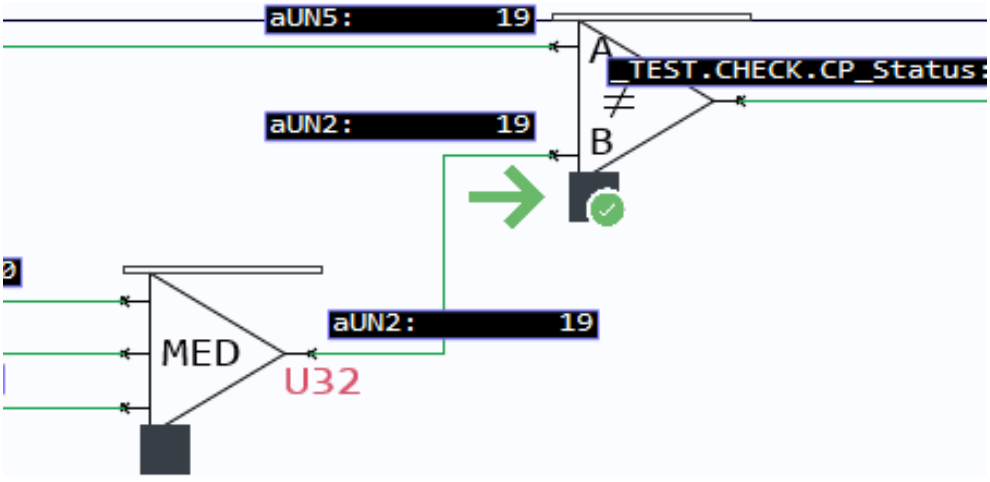


2. Click **OK** button.

   Active breakpoint figure



About Breakpoints and Debugger Tool buttons



User Interface

The preceding figure shows an application with several ordinary components without breakpoints, a component with a Set (marked but inactive) breakpoint, and a component with an Active breakpoint.

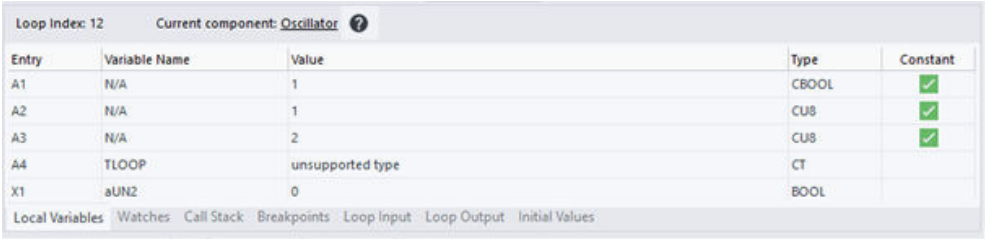
The following table describes how the Debugger buttons work with breakpoints.

Debugger Tool elements description

Icon	Button / window Name	Description
	Current component	Marks the component to which you have stepped the Debugger Tool. This component also appears on top of the current call stack.
	Component in current call stack	Marks a component that is part of the call stack but that is not on top of the call stack.
	Set breakpoint	Indicates a **Set** breakpoint. A Set breakpoint marks a breakpoint of interest but does not interact with the Run, Step Over, or Step Loop buttons. The Debug window's Breakpoints tab lists all the Set and Active breakpoints in an application. Click the **Active** checkboxes in this tab to: • Turn Set breakpoints into Active breakpoints. • Turn Active breakpoints into Set breakpoints.
	Active breakpoint	Indicates an **Active** breakpoint. An Active breakpoint interacts with the Run, Step Over, or Step Loop buttons. The Debug window's Breakpoints tab lists all the Set and Active breakpoints in an application. Click the **Active** checkboxes in this tab to: • Turn Set breakpoints into Active breakpoints. • Turn Active breakpoints into Set breakpoints.
	Run	Runs the Debugger Tool in a non-stop cyclic loop. However, an Active breakpoint in the application interrupts this command if and when its associated component is reached. An Active Data Breakpoint also interrupts this command if and when the associated watch data value is modified. As needed, use the Pause button to manually interrupt this command or use the Stop button to completely end the Debugger Tool session.
	Step	Steps the Debugger Tool to the next component to be executed in the application. An Active Data Breakpoint can interrupt this command if the associated watch data value is modified by the Current component.
	Step Over	Steps the Debugger Tool to the next component (that is not located in a subpage of the current page) to be executed in the application. However, an Active breakpoint located in a subpage interrupts this command if and when its associated component is reached. An Active Data Breakpoint also interrupts this command if and when the associated watch data value is modified.
	Step Loop	Steps the Debugger Tool to the first component to be executed in the application. However, an Active breakpoint located between the current component and the first component in the execution order interrupts this command if and when its associated component is reached. An Active Data Breakpoint also interrupts this command if and when the associated watch data value is modified.

User Interface

Debug Window



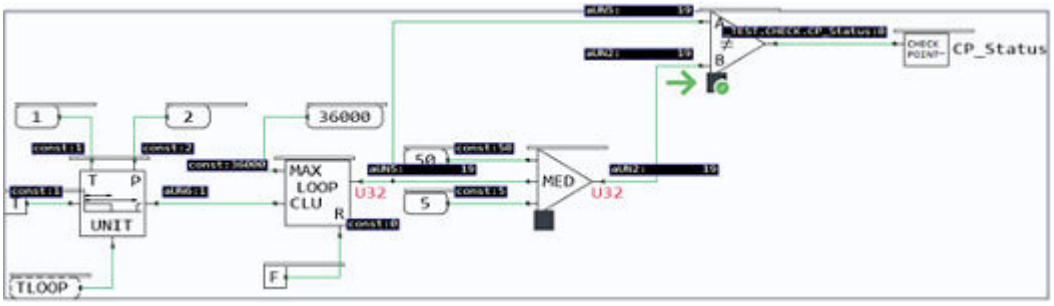
Debug window tabs description

Tab	Description
Local Variables	Use this tab to view the input and output values of the current component or PLC POU to which you have stepped the Debugger tool.
Watches	Use this tab to track signals value of current interest and to create Data Breakpoints that trigger when the values of watched variables change.
Call Stack	Use this tab to see where you have stepped the Debugger Tool in the application hierarchy.
Breakpoints	Use this tab tab to manage and create Set and Active breakpoints in your application. You can use this tab to create an Active breakpoint, switch a breakpoint between Set and Active, or make a conditional breakpoint.
Loop Input	Use this tab to input values to an application at the start of program loops. The GUIDE application saves these values from your Debugger Tool session for future use.
Loop Output	Use this tab to record the outputs of program loops.

User Interface

Debug Window—Local Variables tab

Use the **Local Variables** tab to view the input and output values of the current component or PLC POU to which you have stepped the Debugger Tool.



Loop Index: 12 Current component: Oscillator

Entry	Variable Name	Value	Type	Constant
A1	N/A	1	CBOOL	✓
A2	N/A	1	CU8	✓
A3	N/A	2	CU8	✓
A4	TLOOP	unsupported type	CT	
X1	aUN2	0	BOOL	

Local Variables Watches Call Stack Breakpoints Loop Input Loop Output Initial Values

Local Variables tabs description

		Description
→	Current Component icon	Marks the current component.
MED	Component picture	The current component as shown in the Contents chapter of the PLUS+1 GUIDE User Manual. The labels in the picture correspond with items in the Entry column of the Local Variables tab.
	Loop Index	The current program loop. The Loop Index count starts at 0.
	Current component	The component to which you have stepped the Debugger Tool. Click the underlined name of the component to center it in the Drawing Area.
?	Help	Click to view the description of the **Current component**.
	Local Variables tab	Lists the local variables of the **Current component** or PLC POU. • **Entry** column — lists the entries of the nets connected to the Current component. Items in this column correspond with the labels shown in the picture of the component in the PLUS+1 GUIDE User Manual. • **Variable Name** column — lists the net variable name or the PLC variable name. Constants do not have variable names. • **Value** column — lists the value of the nets that connect to the current component. Left-click a Value column cell to change the cell value. A changed value takes effect immediately. Note that changed output values usually (depending on the type of component) get overwritten as soon as the component executes. • **Type** column — lists the Data Type of each net connected to the current component. • **Constant** column — checkmarks identify nets with constant values.

User Interface

Debug Window—Watches tab

Use the **Watches** tab to keep track of a customizable set of signal values, optionally to create Data Breakpoints for some of these signal values that trigger when their values change.

While there is no limit on the number of concurrently active Watches, there is a limit of up to four concurrently Active Data Breakpoints.

If you have a watched variable with an enabled Data Breakpoint, the Debugger Tool:

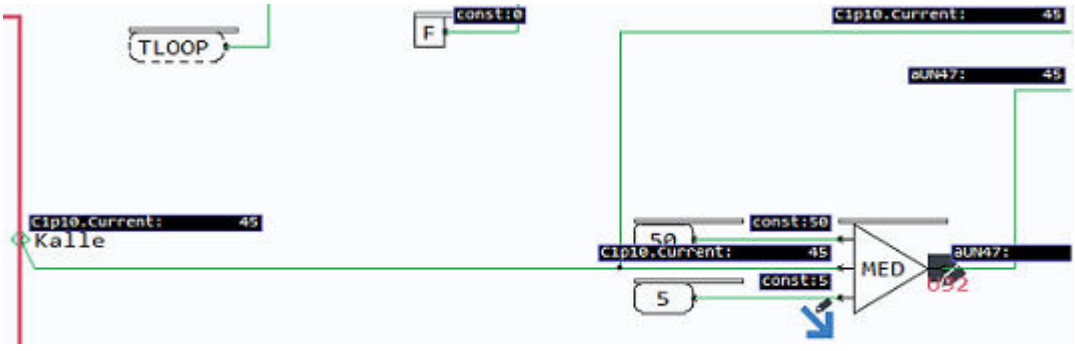
- Displays a Current Data Breakpoint icon at the component whose watched variable has changed in value.
- Displays a Data Breakpoint icon where the watched variable changed value.

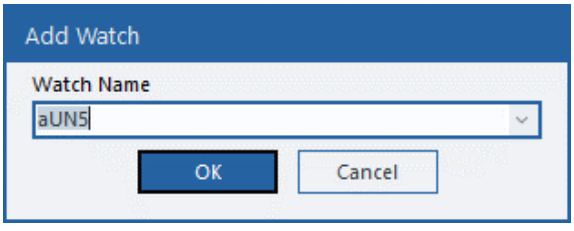
Note that only data that changes within the application while the Debugger Tools runs will trigger a Data Breakpoint. Data that is changed manually by the user, or automatically as part of Loop Input data setting, will not trigger Data Breakpoints.

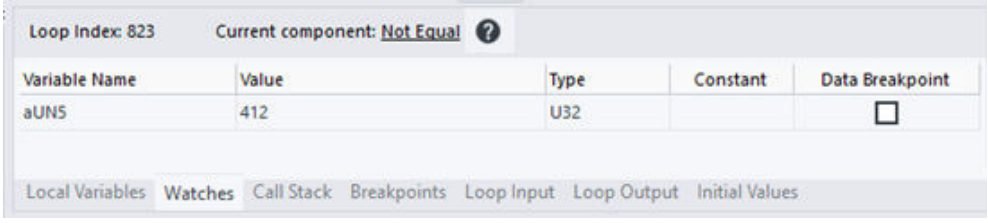
Timing differences between normal breakpoints and Data Breakpoints:

- A normal breakpoint breaks before its component executes and its output changes.
- A Data Breakpoint breaks when the watched output of the component changes. The changed output may be the result of a full or a partial execution of the component.

Complex components may have a mixture of outputs based on the current loop and previous loops. However, a watched output with a Data Breakpoint always shows results from the current loop.







User Interface

Figure and Watches tab description

	Description
Watched variable with a Data Breakpoint Identifier icon	The watched variable and its current value. The Watches tab lists this variable and its value in the tab's **Variable Name** and **Value** columns. To place a Data Breakpoint on a variable that you want to watch either: • Right-click in the **Watches** tab to open the **Add Watch** window. Type the name of the variable to watch. • • Or right-click in the **Local Variables** tab on the row with the variable to watch. In the pop-up menu that opens, click the **Add to Watch List** command. See *Debug Window—Local Variables tab* on page 156 for more information. Click the **Variable Name** to display the Data Breakpoint Identifier icon and to center the Drawing Area on the icon.
Current Data Breakpoint icon	When the value of a watched variable changes, the Debugger Tool displays the Current Data Breakpoint icon at the component whose watched variable has changed in value.
Watched variable with a Data Breakpoint icon	The watched variable and its changed value. The Watches tab shows the changed value of this variable in its **Value** column. When the value of a watched variable changes, the Debugger Tool displays a Data Breakpoint icon where the watched variable changed value.
Loop Index	The current program loop. The Loop Index count starts at 0.
Current component	The component to which you have stepped the Debugger Tool. Click the underlined name of the component to center it in the Drawing Area.
Help	Click to view the description of the **Current component**.
Watches tab	• **Variable Name** column — lists the names of the watched variables. • **Value** column — lists the value of the watched variable. A Data Breakpoint triggers when this value is changed by the application while executed by the debugger if you have clicked the **Data Breakpoint** checkbox. Loop Input data and manual data modifications will not trigger Data Breakpoints. You can directly enter values to watch in this column. • **Type** column — lists the Data Type of the watched variable. • **Constant** column — checkmarks make watches active. • **WatchId** - lists the interface names of the watched variables.

User Interface

Debug Window—Call Stack tab

Use the **Call Stack** tab to see where you have stepped the Debugger Tool in the application hierarchy.

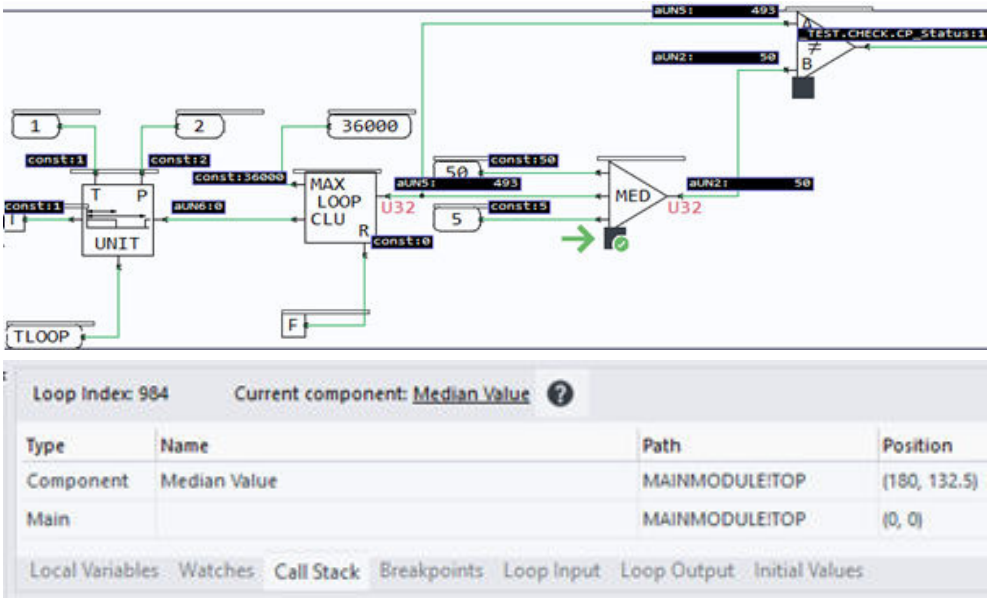


Figure and Call Stack tab description

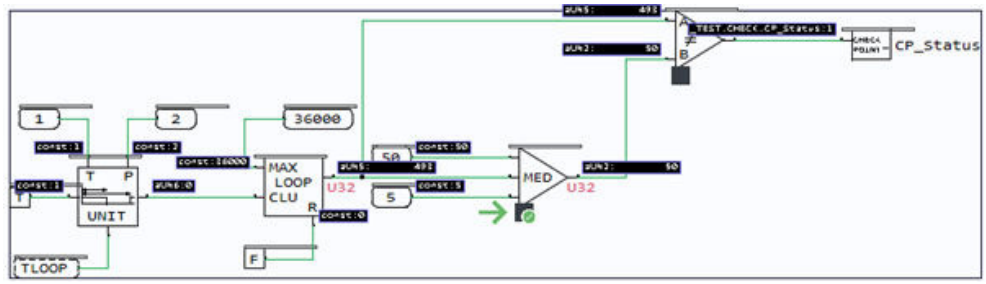
	Description
Current Component icon	Marks the current component.
Loop Index	The current program loop. The Loop Index count starts at 0.
Current component	The component to which you have stepped the Debugger Tool. Click the underlined name of the component to center it in the Drawing Area.
Help	Click to view the description of the **Current component**
Call Stack tab	• **Type** column — lists the type of application building block. • **Name** column — lists the name of the Current component or POU. Double-click to center the Drawing Area on the Current Component. • **Path** column — lists the path to the Current component or POU. • **Position** column — lists the position of the Current component or the line number in the PLC code or the Local ID of an FBD/LD component.

User Interface

Debug Window—Breakpoints tab

Use the **Breakpoints** tab to manage breakpoints in your application. You can:

- Make **Set** breakpoints **Active** and return Active breakpoints to their Set condition.
- Create Conditional breakpoints that trigger when conditions that you have defined become true.
- Add Active breakpoints to this tab and delete breakpoints listed in this tab.



Debug window tabs description

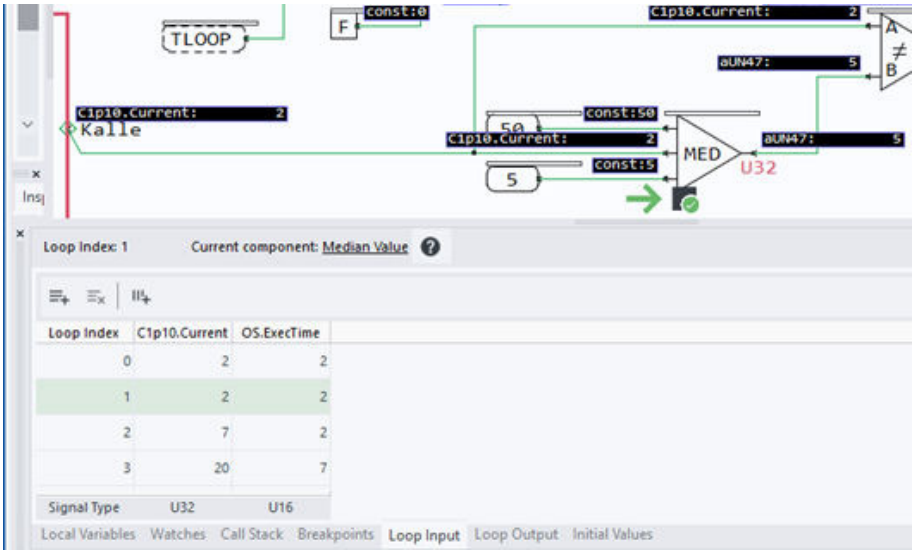
	Description
Set breakpoint	Check **Active** checkboxes in the Breakpoint tab's Active column to make breakpoints Active.
Current Component icon	When a Conditional breakpoint becomes true, it can interrupt execution like a normal **Active** breakpoint. In the preceding figure, the **Condition** set for the **Not Equal** component's breakpoint becomes true when **aN183 == 800.**
Conditional Breakpoint icon	Indicates a Conditional breakpoint. To create a Conditional breakpoint: • Check the breakpoint's Active checkbox in the tab's Active column. • Enter a **Condition** to trigger the breakpoint in the tab's Condition column. Express the condition as a C language Boolean expression.
Active breakpoint	Uncheck Active checkboxes in the Breakpoint tab's Active column to return Active breakpoints to a Set condition.
Loop Index	The current program loop. The Loop Index count starts at 0.
Current component	The component to which you have stepped the Debugger Tool. Click the underlined name of the component to center it in the Drawing Area.
Help	Click to view the description of the **Current component**.
Breakpoints tab	• **Active** column — click checkboxes to make Set breakpoints Active and to return Active breakpoints to their Set condition. • **Name** column — lists all the components in the application that have Set and Active breakpoints. • **Path** column — lists the the path to the breakpoint. Click to go to the page with the breakpoint. • **Position** column — lists the x and y axis locations of the breakpoint. Click to center the breakpoint in the Drawing Area. • **Condition** column — lists the condition that when true triggers an Active breakpoint.

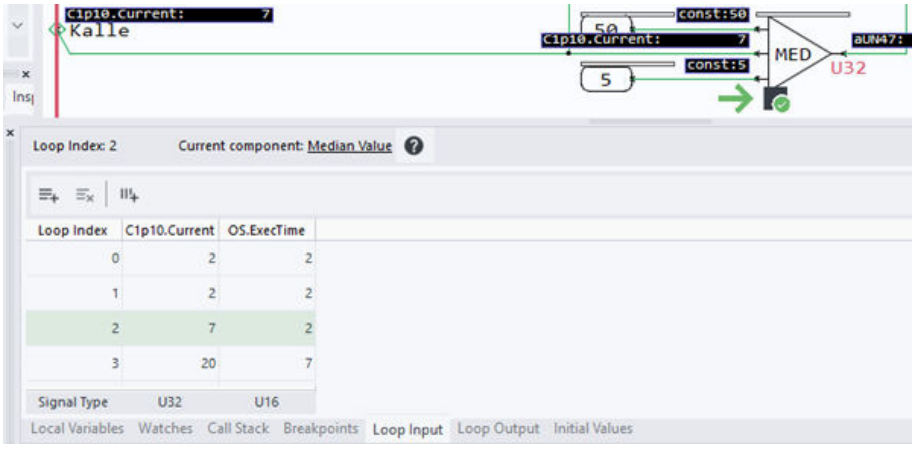
User Interface

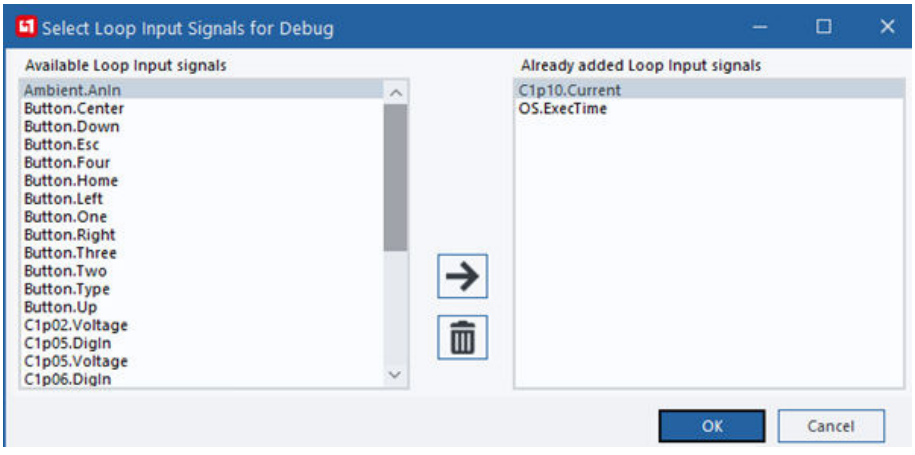
Debug Window—Loop Input tab

Use the **Loop Input** tab to input values to an application at the start of program loops. The GUIDE application saves the values that you enter in this tab as part of the Debugger Tool file.

By default, the **Loop Input** tab opens with only the **Loop Index** column displayed. Right-click to open the **Select Loop Input Signals for Debug** window. Use this window to create columns for the signals that you want to input to the application.







User Interface

Debug window tabs description

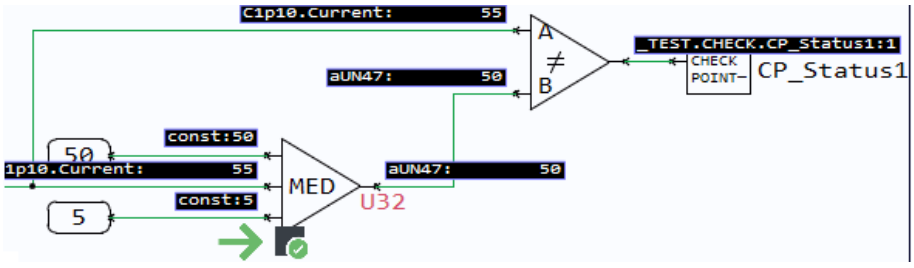
	Description
Loop Index	The current program loop. The Loop Index count starts at 0.
Current component	The component to which you have stepped the Debugger Tool. Click the underlined name of the component to center it in the Drawing Area.
Help	Click to view the description of the **Current component**.
Loop Input tab	The highlighted bar moves down as you loop through the application. Columns list controller inputs. Enter the values that you want to apply in these columns.
Pop-up menu	Right-click in the **Loop Input** tab to open a pop-up menu. Use the commands in this menu to: • Edit the **Loop Input** table. • Open the **Select Loop Input Signals for Debug** window. This window lists all valid controller input signals. Use the Select Loop Input Signals for Debug window to create columns in the **Loop Input** tab for the signals that you want to input to the application.

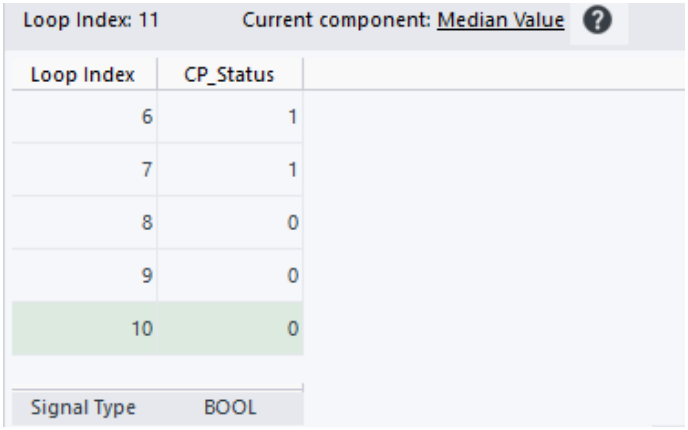
User Interface

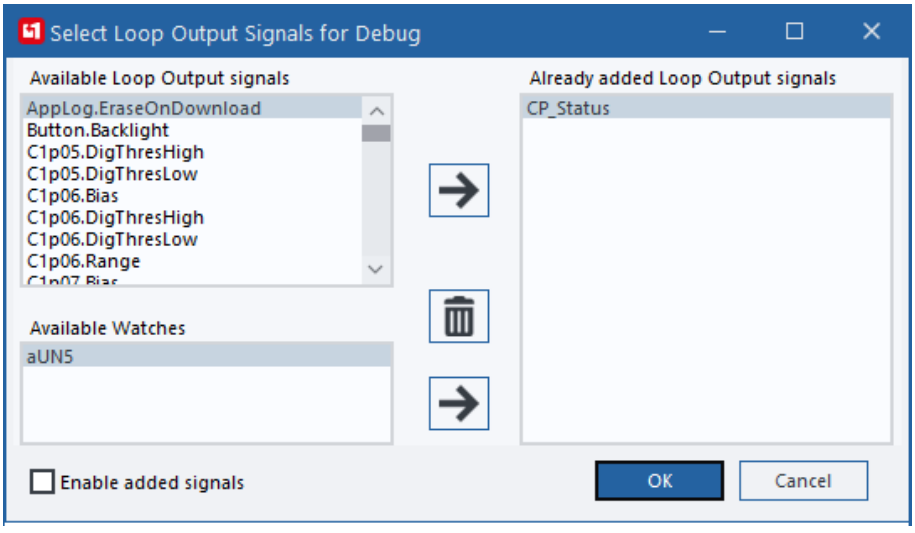
Debug Window—Loop Output tab

Use the **Loop Output** tab to record the outputs of program loops.

By default, the **Loop Output** tab opens with only the **Loop Index** column displayed. Right-click to open the **Select Loop Output Signals for Debug** window. Use this window to create columns for the signals that you want to output from the application.







Debug window tabs description

	Description
Loop Index	The current program loop. The Loop Index count starts at 0.
Current component	The component to which you have stepped the Debugger Tool. Click the underlined name of the component to center it in the Drawing Area.
Help	Click to view the description of the **Current component**.

User Interface

Debug window tabs description (continued)

	Description
Loop Output tab	The highlighted bar moves down the rows as you execute program loops. • **Loop Index**column — the current program loop. • Loop output columns—record the outputs of program loops.
Pop-up menu	Right-click in the **Loop Output** tab to open a pop-up menu. Use the commands in this menu to: • Enable or disable the output of signal values to the **Loop Output** tab. • Export values from the Loop Output tab to a CSV format file. • Open the **Select Loop Output Signals for Debug** window. This window lists all valid output signals. Use the Select Loop Output Signals for Debug window to create columns in the **Loop Out** tab for both output and **Watches** signals from the application. Click the **Enable** added signals checkbox to enable the output of all the signals that you add to the **Loop Output** tab.

Debugger skin

If the debugged application is a display application, with a properly created skin, the skin will be displayed with buttons and LEDs. The buttons can be pushed and the LEDs can be lit.

Virtual CAN Gateway

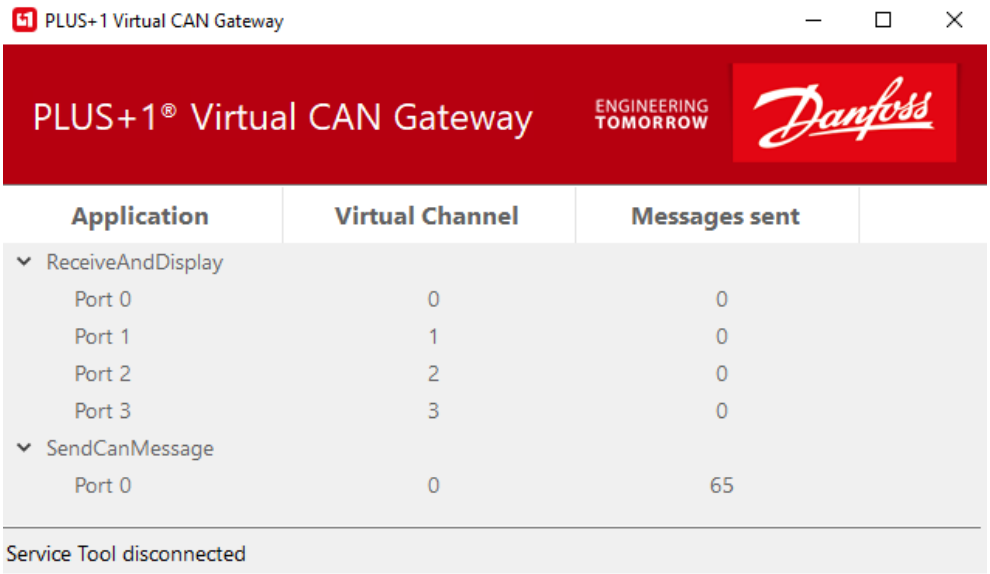
The PLUS+1 Virtual CAN Gateway enables to virtually interconnect a network of debugged applications. Use the PLUS+1 Virtual CAN Gateway to test a complete system without any physical hardware.

The PLUS+1 Virtual CAN Gateway receives the incoming CAN messages and forwards them to the other connected applications. The function starts when the Debugger Tool is started. The debugged application automatically connects to the PLUS+1 Virtual CAN Gateway. Only one debugged application can be connected to the PLUS+1 Virtual CAN Gateway via a debugging session in PLUS+1 GUIDE, others must be connected standalone.

To open the user interface, click on the PLUS+1 Virtual CAN Gateway icon in the system tray, in the taskbar.

User Interface

User Interface



Column	Description
Application	Lists all the debugged applications, and their corresponding ports, that are connected to the PLUS+1 Virtual CAN Gateway.
Virtual Channel	Indicates which virtual channel each port is routed to. This value is configurable by double clicking on the value, change to a number between 0-4.
Messages sent	Indicates the number of CAN messages that have been sent by each port.

Generate FMU

This is a PLUS+1® GUIDE upgrade feature. A Quality Assurance License (reference PLUS+1® GUIDE add on license Quality Assurance License Data Sheet, AI170686484256) enables this feature.

For more information, see PLUS+1® GUIDE Licensing, **AQ419969404812**.

Use the Generate FMU functionality to export a PLUS+1® GUIDE page or module to a Functional Mock-up Unit (FMU) according to version 2.0.1 of the Functional Mock-up Interface standard. Refer to *https://fmi-standard.org/*.

Generated FMUs support both Model Exchange and Co-Simulation.

Limitations

Values of data types U32 and U64 at the FMU interface are limited to the range [0..2147483647].

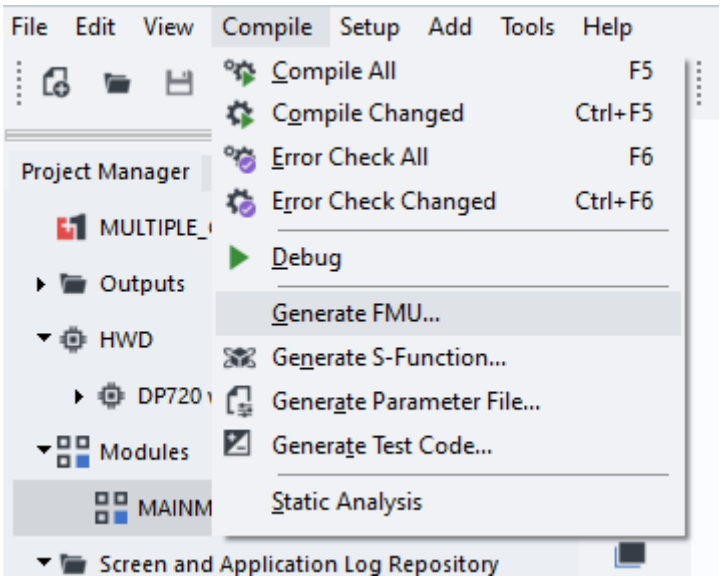
Values of data type S64 at the FMU interface are limited to the range [-2147483648..2147483647].

How to generate an FMU

1. Navigate to the page to simulate.

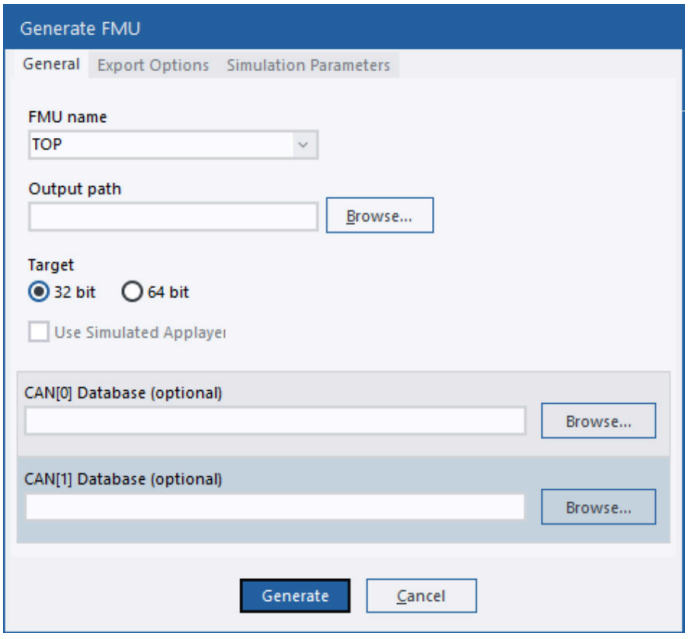
User Interface

2. Select **Compile** > **Generate FMU**.



3. Use the **Generate FMU** dialog to setup the FMU export.

   There is one tab for general settings, one for export options and one for simulated applayer settings (requires HWD support). Each of the tabs are described below.



General settings

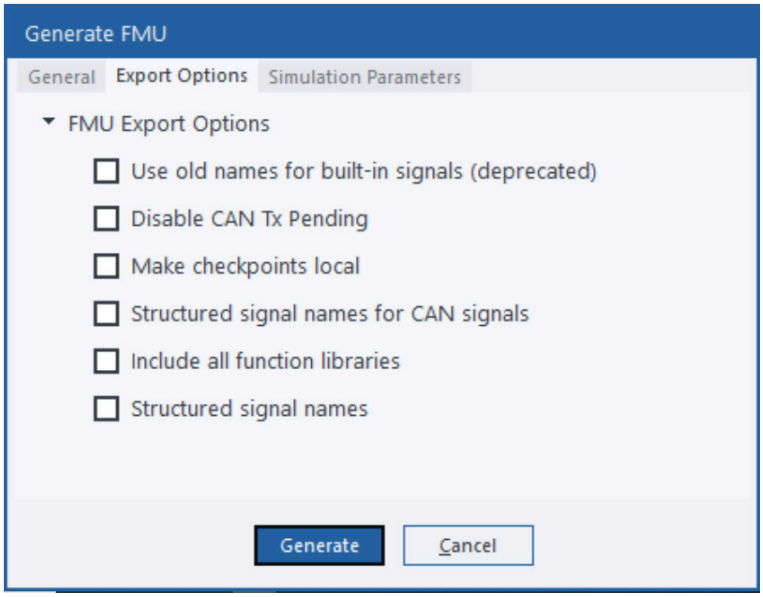
Name	Description
FMU name	Set the name of the FMU. It can be typed in or set either to the name of the page to simulate or to the most recently entered name, if any.
Output path	Choose target folder for the generated FMU. It can be set to an absolute path or a path relative to the project folder. Leave empty to generate the FMU into the project folder. Use the Browse button to open a folder dialog.
Target	Set the architecture of the generated FMU. Valid options are 32-bit or 64-bit.

User Interface

General settings (continued)

Name	Description
Use Simulated Applayer	When checked, BIOS signals may be simulated in the FMU instead of appearing at the FMU interface. See section *Simulated applayer* on page 168. Requires HWD support.
CAN Database	Specify a CAN database file to use. CAN messages in the CAN database will appear at the interface of the generated FMU. Refer to PLUS+1® Service Tool Design Manual for more information on CAN databases. Each CAN port has its own CAN database.

Export options

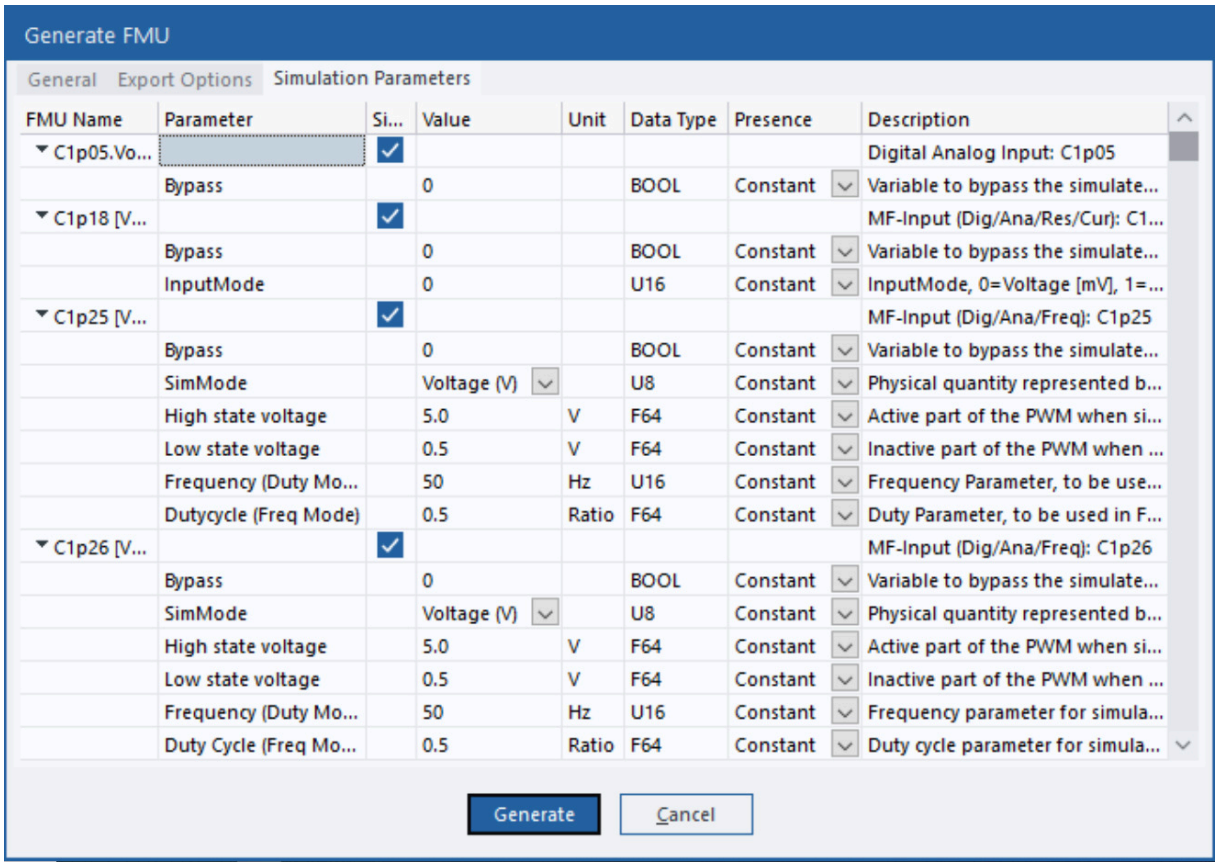


Export options

Name	Description
Make checkpoints local	Checkpoint signals will have their causality set to local in the FMU interface. Using this option may reduce the number of outputs shown in the simulation tool.
Include all function libraries	Include all function libraries in the project when generating the FMU. Do not use this option unless there is a compile problem related to library dependencies.
Disable CAN Tx Pending	The pending output of Transmit CAN components will always be low. If this option is false, the pending output of transmit CAN components will be true during the program loop after the transmission. For further information, see *Transmit CAN* on page 325.
Use old names for built-in signals (deprecated)	Interface signals OS.ExecTime_CoSimulation_Input and Default_CAN_bit_value were called “OS.ExecTime(CoSimulation_Only)” and “Default CAN bit value” in GUIDE versions prior to 2022.3. Only use this option for existing simulation environments.
Structured signal names for CAN signals	Separators in CAN signal names will get names suitable for structured naming convention. It is described in detail in chapter *CAN Database* on page 170.
Structured signal names	Use structured naming convention in the FMU interface. To support array and string signals in the FMU interface, this option must be checked.

User Interface

Simulated applayer



FMU Name	Parameter	Si...	Value	Unit	Data Type	Presence	Description
C1p05.Vo...		✓					Digital Analog Input: C1p05
	Bypass		0		BOOL	Constant	Variable to bypass the simulate...
C1p18 [V...		✓					MF-Input (Dig/Ana/Res/Cur): C1...
	Bypass		0		BOOL	Constant	Variable to bypass the simulate...
	InputMode		0		U16	Constant	InputMode, 0=Voltage [mV], 1=...
C1p25 [V...		✓					MF-Input (Dig/Ana/Freq): C1p25
	Bypass		0		BOOL	Constant	Variable to bypass the simulate...
	SimMode		Voltage (V)		U8	Constant	Physical quantity represented b...
	High state voltage		5.0	V	F64	Constant	Active part of the PWM when si...
	Low state voltage		0.5	V	F64	Constant	Inactive part of the PWM when ...
	Frequency (Duty Mo...		50	Hz	U16	Constant	Frequency Parameter, to be use...
	Dutycycle (Freq Mode)		0.5	Ratio	F64	Constant	Duty Parameter, to be used in F...
C1p26 [V...		✓					MF-Input (Dig/Ana/Freq): C1p26
	Bypass		0		BOOL	Constant	Variable to bypass the simulate...
	SimMode		Voltage (V)		U8	Constant	Physical quantity represented b...
	High state voltage		5.0	V	F64	Constant	Active part of the PWM when si...
	Low state voltage		0.5	V	F64	Constant	Inactive part of the PWM when ...
	Frequency (Duty Mo...		50	Hz	U16	Constant	Frequency parameter for simula...
	Duty Cycle (Freq Mo...		0.5	Ratio	F64	Constant	Duty cycle parameter for simula...

Bios pins may be simulated by the FMU when simulated applayer is enabled. For example, a multi-functional input pin has many inputs: Phase, Period, Frequency, Duty cycle, Voltage, Digital input etc. If simulated applayer is not enabled, these signals will end up in the FMU interface where they will need to be handled in the simulation tool.

All these signals will be handled internally by the FMU when simulated applayer is enabled. Instead of exposing all bios signals related to a certain pin in the FMU interface, there will be a signal representing the pin itself. Simulated pins have a set of configuration entities. Each entity can be set as a constant, an FMU parameter or an FMU signal.

Simulated applayer requires HWD support.

Generated FMU Properties

The interface of the generated FMU will consist of the page interface of the simulated page, bios signals, module connections and diagnostic signals. There will also be two signals to control the simulation.

Co-Simulation step size

In addition to the above mentioned interface signals, a signal named "OS.ExecTime(CoSimulation_Only)" will always be present. Use it to set the step size in milliseconds during Co-Simulation. As indicated by the name, it will not have any purpose for Model Exchange.

If "OS.ExecTime(CoSimulation_Only)" is not set, the value assigned to OS.ExecTimeOut will be used if it is set.

Step size will be set to 1 ms if none of the signals described above are set.

User Interface

OS Signals

Bios signals in the OS struct are handled in two different ways:

OS.Start, OS.LoopCnt, OS.ETime will be simulated. OS.ExecTime will be set to the step size of the simulation.

All other members will be added as inputs or outputs to the FMU.

Non-volatile memory

Non-volatile memory signals have causality set to parameter and variability set to tunable.

Set Pulse

The data type of Set Pulse signals is set to GUIDE.TYPE.PULSE which is a Boolean data type. This will make it possible to distinguish it from other interface signals.

Repeat/Until

A repeat/until loop in a PLUS+1® GUIDE module will time out if (number of loops + number of executed Get Time µs components) > (OS.ExecTime * 1000).

Unique signal names

All names in the FMU interface must be unique. Interface signals are sometimes renamed to ensure this.

Page Interface

In case of a naming collision, signals will be renamed. Signals connected directly to a page will get the prefix “S_”. Signals in buses will get the prefix “Bη_”, where η is the bus depth (a signal in a bus connected to the page interface has a depth of 1, a signal in a bus in a bus connected to the page interface has a depth of 2 and so on).

If a renamed signal happens to get the same name as another signal, it will get the additional prefix “Renamedη_”, where η is a number that will make the renamed signal name unique.

Value Connect and capped components

Signals connected to the output of Value Connect or Value Initialization or the second output of capped components may be connected to other outputs in the same category. Since they may be both read and written, they will appear both as inputs and outputs if they are connected to the interface of an FMU. The input signal will get the prefix “_connect_in_”, the output signal will get the prefix “_connect_out_”.

Bidirectional bios signals

Bidirectional bios signals can be both read and written. In the FMU interface, each bidirectional signal will be added as an input with the prefix “_input_” and an output with the prefix “_output_”.

Arrays and strings

Export option **Structured signal names** must be checked to support array and string signals in the FMU interface.

There will be one interface signal for each element in an array. For example, if checkpoint A is connected to an array of U8 of length two, the following signals will appear in the interface:

A[0]
A[1]

Strings work in a similar way. If a string has no explicit length, it will get as many signals as the default string length in the GUIDE application. Each signal will be of type CHAR which is an unsigned 8-bit type. If the length of a string is less than its maximum length, it will be indicated by a null termination. For example, if string B has a length of three and a value of “A”, its signals will have the following values:

B[0] = 65 (UTF8 A)

B[1] = 0 (null termination)
B[2] = ? (undefined, since the length is one)

All array and string indices start at zero.

Simulated CAN interface

CAN Database

Signals in the CAN database will be added to the interface of the FMU. Their names will depend on the export option **Structured signal names for CAN signals**:

Structured signal names for CAN signals state	Naming format
Checked	CANx.Message.Signal
Unchecked	CANx->Message->Signal

In the table above, x is the CAN channel.

All signals in messages with the TxNode attribute will be added as outputs from the FMU. Whenever a CAN message is sent within the FMU, its interface signals will be updated.

The signals in messages without the TxNode attribute will be added as inputs to the FMU. If there is no Traffic node specified for a given input signal, its message will be received each program loop. When a Traffic node is specified, the behavior depends on the Type attribute:

Cyclic The CAN message will be received at an interval specified by the Rate attribute.

OnChange A CAN message will be received if any of its signals has been assigned a different value since last program loop.

OnWrite A CAN message will be received if any of its signals has been written since last program loop.

Unsupported features

The information in this chapter applies to PLUS+1® GUIDE 12.1 and may be subject to change in later versions.

Within the Signal node, the following elements are unsupported or have no meaning:

- Scale
- Range
- Info
- RxNode
- Condition

Attribute DataSource in element Decode is also unused. However, if the value of DataSource is “CANid”, the signal containing the DataSource element will be ignored.

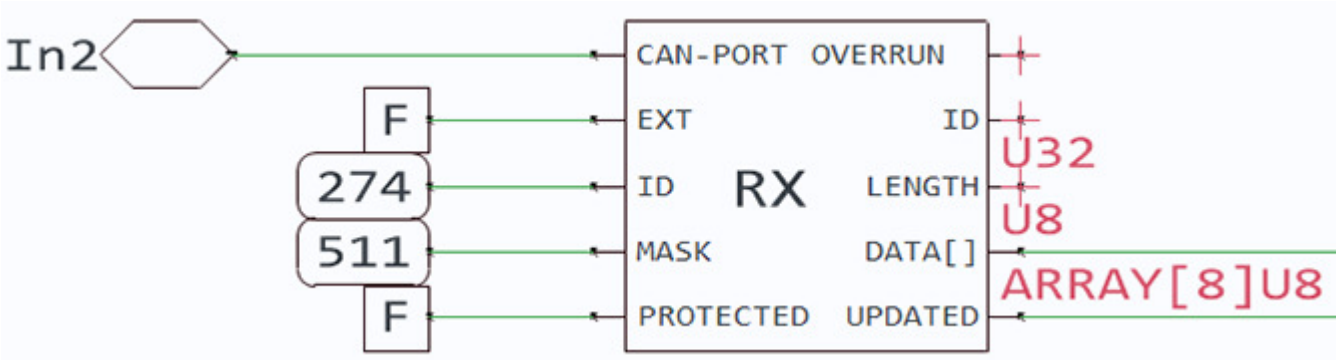
Network and Node elements are ignored. (At least one Network element must exist in the CAN database, or else no messages can be specified.)

Select CAN port

Depending on where the CAN[x].Port signal is defined, the port may have to be specified when running the simulation.

User Interface

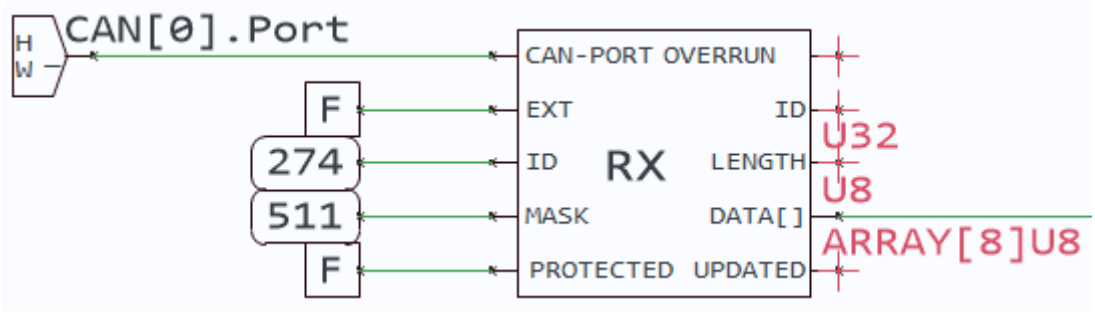
CAN port is defined outside the simulated scope; it needs to be set in the FMU interface



In the image above, the port must be specified in the simulation. The FMU interface will contain a signal In2 of type port. Set it to 0 for CAN[0].Port, 1 for CAN[1].Port and so on.

The PORT type in the FMU interface is different from the PORT type in PLUS+1® GUIDE. The only supported use of data type PORT in the FMU context is to select CAN port. In the FMU interface, data type PORT is an integer in the interval [0..15].

CAN port is defined inside the simulated scope – no action is needed



In addition to the OS signals mentioned above, CAN[x].Port will be handled automatically and not propagated to the FMU interface.

If the CAN-PORT input of a CAN component has not been assigned it will not be considered when CAN messages are processed (this will happen if the PORT signal must pass through loop delays or if it is processed after the CAN component).

No CAN messages will be received during the first program loop.

CAN callback functions

Callback functions can be passed to an FMU to control reception of custom CAN messages and to log all CAN messages sent by the FMU, bypassing the CAN database functionality. To manage callback functions, the following functions are exported by the FMU DLLs:

- void AddCANReceiveCallback(void *Context, CANReceive_Fcn_T CANReceiveCallback);
- void RemoveCANReceiveCallback(CANReceive_Fcn_T CANReceiveCallback)
- void AddCANSendCallback(void *Context, CANSend_Fcn_T CANSendCallback);
- void RemoveCANSendCallback(CANSend_Fcn_T CANSendCallback);

A context pointer and a callback function are passed to the two add functions to register or update a callback. Whenever a callback is called, the context pointer passed to the add function together with that callback will be provided. Passing the same callback twice to an add function will update the associated context pointer, but the callback will not be registered twice.

The two remove functions will remove the callback if it is registered, otherwise they will do nothing.

User Interface

The types of the callback functions are defined as below:

```
typedef int32_t __stdcall (*CANSend_Fcn_T)(
                              void *const Context,
                              uint8_t port,
                              uint32_t Identifier,
                              uint32_t DLC,
                              uint8_t *Data,
                              uint32_t Flags);
```

All CAN send callbacks registered in an FMU will be called whenever a CAN message is sent by that FMU.

Argument	Description
Context	The context pointer associated with the callback. Use it to identify the calling FMU.
port	The CAN port where the CAN message was sent.
Identifier	ID of the CAN message.
DLC	CAN message length.
Data	Array of data bytes.
Flags	CAN message flags. ext=bit0, rtr=bit1

```
typedef int32_t __stdcall (*CANReceive_Fcn_T)(
                              void *const Context,
                              uint8_t *port,
                              uint32_t *IdentifierOut,
                              uint32_t *DLCOut,
                              uint8_t *DataOut,
                              uint32_t DataOutSize,
                              uint32_t *FlagsOut);
```

All CAN receive callbacks registered in an FMU will be called before that FMU runs the program loop. When a CAN receive callback returns a non-zero value, a CAN message will be received by the FMU according to the parameters of the callback as described in the table below. Then the callback will be called again. If it returns zero, no CAN message will be received, and the callback will not be called again until the next program loop.

Argument	Description
Context	The context pointer associated with the callback. Use it to identify the calling FMU.
port	Set it to the CAN port where the CAN message was sent.
IdentifierOut	Set it to the ID of the CAN message.
DLCOut	Set it to the CAN message length.
DataOut	Set it to the Array of data bytes.
DataOutSize	Length of DataOut.
FlagsOut	Set it to the CAN message flags. ext=bit0, rtr=bit1

Set default bit value

Use the FMU interface signal “Default CAN bit value” to select the default value of data bits in a CAN message. The default value will be 0 if “Default CAN bit value” is set to 0 and 1 otherwise. When a CAN message is received in the FMU, undefined bits will have the default value.

File Types

PLUS+1® GUIDE file types are listed, excludes common file types and temporary/internal file types.

PLUS+1 GUIDE File Types

The following table excludes common file types and temporary/internal file types.

PLUS+1® GUIDE file types

File Type	Description	Comment
P1X	PLUS+1® GUIDE Project file	
P1P	Packed PLUS+1® GUIDE Project file	Note: Do not pack to P1P if you want to use version control for your PLUS+1® Projects. *(There is a setting in the Options dialog to disable the P1P packing dialog on project close.)* See also the **Filemenu** action: " Export project as 7z/Zip …"
SCS	A module containing graphical code in PLUS+1® GUIDE	File name must start with a capital letter, and only contain letter, digit, underscore, space and hyphen characters. The file name should not contain more than one consecutive space character, and should not end in a space character.
HWD	Hardware description file	These files are installed in PLUS+1® GUIDE, and contain all information needed to develop for a specific PLUS+1® Hardware type.
SYS	System file	This file is the part of the HWD file that is needed to compile for a specific PLUS+1® Hardware type. It is transferred to the projects targeting this Hardware, and contains object code that is linked into the resulting downloadable LHX file.
LHX	Downloadable PLUS+1® application file	Use PLUS+1® Service Tool (or another compatible tool) to download this file.
P1D	PLUS+1® Service Tool Application file	This file type is normally created from within the Service Tool, but can also be automatically generated from PLUS+1® GUIDE on compile. When generated from PLUS+1® GUIDE, it is intended for development support, not for use in the field or in production.
EXR (Read only parameters)	Read only parameters definition	Generated from CSV file.
P1M	PLUS+1® GUIDE Project manifest file	Temporary project file used by GUIDE for internal project handling. • Do not edit. • Do not add to source control.

Programming

Learn more about the specifics of programming in PLUS+1® GUIDE, including the supported PLC programming languages, and how to use hardware templates.

Specifics of supported data types, page layout guidelines, and how to use C code in your applications are also included.

Programming tips are also described to help you use the PLUS+1® GUIDE tool more efficiently.

Using suitable coding guidelines for the programming languages in use is good practice. In safety-related software development, it is mandatory.

Suitable coding guidelines specify good programming practice, proscribe unsafe language features, promote code understandability, facilitate verification and testing, and specify procedures for source code documentation. Coding guidelines serve to ensure good maintainability through the entire application life-cycle.

The following table lists recommended Coding guidelines.

Coding guidelines supported by PLUS+1® GUIDE programming languages

Programming Language	Coding Guidelines
PLUS+1® GUIDE Graphical Code	PLUS+1® GUIDE Development Guidelines
IEC61131-3 PLC Languages	PLCopen Coding Guidelines 2016-04-21 *http://www.plcopen.org/pages/pc2_training/index.htm*
C code in PLUS+1® GUIDE	MISRA C:2012 Coding Guidelines *https://www.misra.org.uk/Buyonline/tabid/58/Default.aspx*

STRING Types

PLUS+1® GUIDE supports the type STRING in GUIDE graphical code, PLC code and in VBSE. The STRING type supports all Unicode Code Points, using UTF-8 encoding.

Within PLC code there is also another type available, WSTRING. WSTRING uses UTF-16 encoding, which is less efficient in most cases. There are exceptions depending on hardware and language used, but in general it is recommended to always use STRING instead of WSTRING.

A variable of STRING type has a current length, a maximum possible length, and several code units equal to its current length. The number of code points in a STRING is always less than or equal to the number of code units in that STRING. STRING types are not null-terminated.

The following table explains these concepts.

STRING type lengths and code units

Concept	Description
Current length	The current number of code units in the STRING.
Maximum length	Each STRING has a finite amount of space reserved to it. A STRING can never become longer than its maximum length. It is possible to set the maximum length of individual STRING variables by specifying their type as STRING[x] instead of just STRING, where x is a number between 1 and 65535. For STRING variables where the maximum length is not given explicitly, the maximum length is a project setting. Use the inspector for the project node in the project manager to adjust this setting. By default, projects will set 255 for STRING when maximum length is not explicitly defined.
Code point	Code point is a value in the Unicode codespace. Simply said, this (usually) corresponds to what is normally thought of as a character.

Programming

STRING type lengths and code units (continued)

Concept	Description
Code unit	In UTF-8, each code point is built up from 1 to 4 code units, and each code unit is 8-bits. In UTF-16, each code point is built up from 1 to 2 code units, and each code unit is 16-bits. A STRING consists of a sequence of 8-bit code units. A WSTRING consists of a sequence of 16-bit code units. In PLC code, a code unit in a STRING is called a CHAR, and a code unit in a WSTRING is called WCHAR. (Neither CHAR nor WCHAR is guaranteed to correspond to a full character). An ASCII character only requires 1 code unit. So STRING that only contains ASCII characters, the current length of the STRING will correspond exactly with the number of code points (characters) in the STRING. All other code points (outside ASCII) will require 2 to 4 code units, each in UTF-8.
Null termination	Some languages, such as C, indicate the end of a character sequence by the character value 0. That is not the case for the GUIDE STRING type, since their lengths are stored in an associated data field. To use STRING variables from GUIDE/PLC code in C code, it is recommended to first convert them from STRING type into null terminated arrays. In principle, code point 0 is valid for use within STRING variables. However, in VBSE it has a special control function. See *How to Specify a STRING Value* on page 175.

How to Specify a STRING Value

The STRING value is written as text between a pair of single quote characters (‘). Most characters can be written as plain text in a STRING, but some special characters must be written using special $-prefixed escape sequences. There is also the possibility to write any character using $-prefixed escape sequences, each with 2 hexadecimal digits representing the code unit (4 hexadecimal digits for WSTRING).

The syntax is the same in both GUIDE graphical code, and in PLC code. However, the PLC editor has support for more Unicode characters. In GUIDE graphical code, it is recommended to use escape sequences for characters above ASCII.

Escape Sequences

Escape sequences example

Character	UTF-8 encoding	Representation in STRING	UTF-16 encoding	Representation in WSTRING
A	0x41	**A** *(Or $41)*	0x0041	**A** *(Or $0041)*
‘	0x27	**$’** *(Or $27)*	0x0027	**‘** *(Or $0027)*
”	0x22	**“** *(Or $22)*	0x0022	**$”** *(Or $0022)*
ß	0xC3 0x9F	**ß** *(Or $C3$9F)*	0x00DF	**ß** *(Or $00DF)*
$	0x24	**$$** *(Or $24)*	0x0024	**$$** *(Or $0024)*
<Newline> (Implementation-independent)	N/A	**$N**	N/A	**$N**
<Line feed>	0x0A	**$L** *(Or $0A)*	0x000A	**$L** *(Or $000A)*

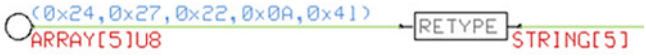
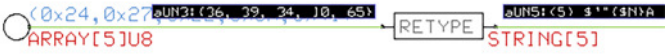
Programming

Escape sequences example (continued)

Character	UTF-8 encoding	Representation in STRING	UTF-16 encoding	Representation in WSTRING
<Carriage return>	0x0D	**$R** *(Or $0D)*	0x000D	**$R** *(Or $000D)*
<Tab>	0x09	**$T** *(Or $09)*	0x0009	**$T** *(Or $0009)*

STRING Examples

STRING examples

Description	PLC example	GUIDE graphical code example
A simple ASCII-only string of length 12 for both STRING and WSTRING: **Hello World!**	My_STRING := 'Hello World!'; My_WSTRING := "Hello World!";	 or 
A simple Latin-1 string of length 2 for STRING, length 1 for WSTRING: **ß**	My_STRING_1 := '**ß**'; //or My_STRING_2 := '**$C3$9F**';	
	My_WSTRING_1 := "**ß**"; //or My_WSTRING_2 := "**$00DF**";	
A somewhat complex string with some special characters, including a line break. Length 5 for both STRING and WSTRING: **$'"** **A**	Complex_STRING := '**$$$'"$NA**'; Complex_WSTRING := "**$$'$"$NA**";	 or  or  or 

Programming

PLC Functions

STRING related PLC functions

Function	Description
Len(string)	Returns the current length of a string. The length of a string is how many Code Units it currently contains, not the number of Code Points/Characters.
Upper_Bound(string)	Returns the maximum length of a string. The maximum length of a string is how many Code Units it can potentially contain, not the number of Code Points/Characters. Using Upper_Bound() for string parameters is an extension of the PLC standard.
Left(string, int-len)	Returns a substring from the start of a string.
Right(string, int-len)	Returns a substring from the end of a string.
Mid(string, int-len, int-pos)	Returns a substring from a specific position within a string.
Concat(string-1, string-2, …)	Returns a combined string of 2 (or more) input strings.
Insert(string-1, string-2, int-pos)	Returns a string where string-2 has been inserted into string-1 at a specific position.
Delete(string, int-num, int-pos)	Returns a string equal to the input string, except for a number of Code Units from a specific position which will be omitted.
Replace(string-1, string-2, int-len, int-pos)	Returns a string equal to string-1, except for a number of Code Units from a specific position which will be replaced by the contents of string-2 instead.
Find(string-1, string-2)	Returns the position of string-2 within string-1, if any. Otherwise returns 0.

VBSE Control Codes

Control code	Meaning
0x00	No characters after (and including) this control character will be printed on the display for the current Text.
0x0A	Continue printing of the current Text on a new line.
0x0D	Ignored.

Using STRING in C Code Files and C Code POUs

To use STRING variables from GUIDE/PLC code in C code, it is recommended to first convert them from STRING type, into null terminated arrays.

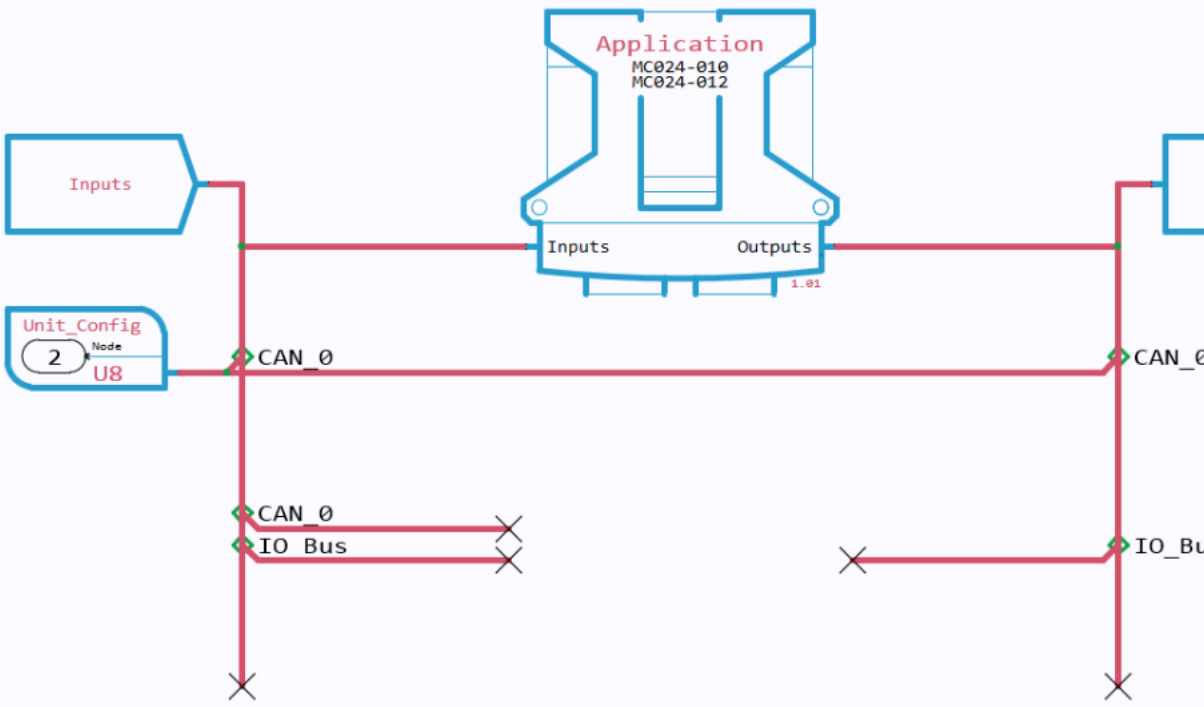
It is technically possible to use the underlying GUIDE data structures directly to manipulate string values in C code, but that is not supported, and those data structures are subject to change from one version of PLUS+1® GUIDE to the next.

Programming

PLUS+1 GUIDE Graphical Code

Hardware Templates

Top level pages in a hardware template



A Hardware Template contains a hierarchy of pages that logically organize application functions. Each PLUS+1® hardware model has its own Hardware Template.

A thin blue line indicates the boundary of the page that you are currently viewing.

A thick light blue line indicates the border of any pages nested within the page that you are currently viewing.

- **Unit Config** page defines the Controller Area Network (CAN) **Node** and **Net**.
- **CAN Config** page sets the CAN baud.
- **Inputs** page has sub-pages that define input functions such as CAN communication, controller pin inputs, and operating system inputs.
- **Application** page contains the page where you create the application for the controller.
- **Outputs** page has sub-pages that define output related functions such as CAN communication, controller pin outputs, and operating system outputs.

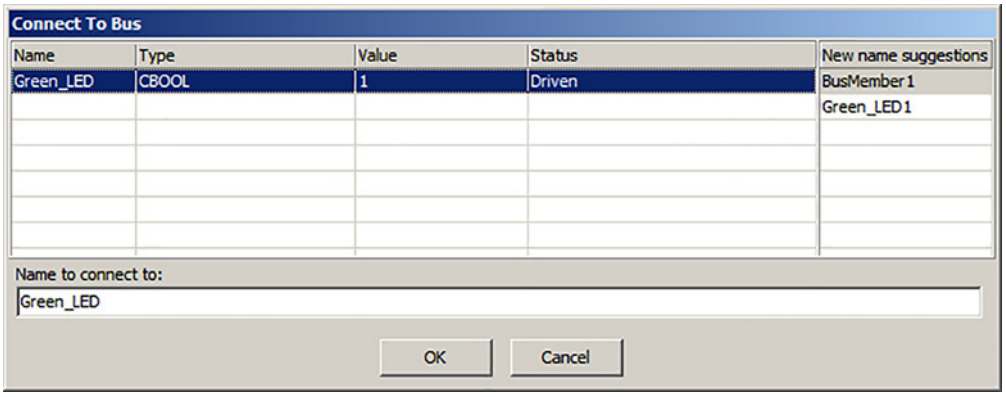
Programming

Route Names

Every route (either a green single signal wire or a red multi-signal bus) in a PLUS+1® GUIDE application must have a name.

The **Connect to Bus** window displays when you connect a route.

Enter your signal names in this window



*Description of **Connect To Bus** dialog*

Item	Description
Name	The name of a signal connected to the bus.
Type	The type of a signal connected to the bus. If no type could be found, this field is left empty.
Value	The value of a constant, or the default value of a variable, connected to the bus. If not constant, or no default value could be found, this field is left empty.
Status	Status is either: **Driven** The wire is connected to an output of a component **Undriven** The wire is not connected to an output of a component **Undriven (with default)** The wire is not connected to an output of a component, but it has a default value.
New name suggestions	A list of names based on previously added names.

When naming routes, observe the following rules:

1. Always begin a signal name with a capital letter.
2. Do not start a signal name with a number.
3. Underscore _ spaces in signal names.
4. Signal names are case-sensitive.
5. A–Z, a–z, and 0–9 are valid signal name characters.

Examples

Rule	Correct name:	Incorrect name:	Result:
1.	Green_LED	green_LED	Compile error
2.	Green_LED_1	1_Green_LED	
3.	Green_LED	Green LED	
4.	Green_LED and GREEN_LED – both are correct names of two different signals.		

Programming

Data Types

- Each data type has a range of values or a specific function, see table below.
- There are overflow conditions that result from using data types whose range of values is too small, for more information go to *About Overflow Conditions* on page 181.
- For more information regarding how the time base data type is used, go to *About the Time Base data type* on page 182.
- For more information regarding the basic characteristics of arrays, go to *About the Array Data Type* on page 184.

Data types used in the PLUS+1® GUIDE software

Data Type	Description	Range
BOOL	True or False	True–False
U8	Unsigned 8 bits	0 to +255
S8	Signed 8 bits	-128 to +127
U16	Unsigned 16 bits	0 to +65535
S16	Signed 16 bits	-32768 to +32767
U32	Unsigned 32 bits	0 to +4294967295
S32	Signed 32 bits	-2147483648 to +2147483647
F32	Floating	$\sim\pm10^{38.53}$
F64	Floating, double precision	$\sim\pm10^{308.25}$
U64	Unsigned 64 bits	0 to +18,446,744,073,709,551,615
S64	Signed 64 bits	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
COLOR	Color used; defined by the operating system	——
FILE	Identifies a window in which graphics display	——
FONT	Font used; defined by the operating system	——
LANG	Sets display language	——
NULL	No connection	——
OBN	Object name	——
PORT	Hardware port; must match hardware port name	——
PXO	Pixel object; pointer to an external graphic	——
T	Time base	——
TL	Text label; pointer to string in text table	——
STRING	An UTF-8 encoded string. Can also be written as: **STRING[x]** where **x** is a number between 1 and 65535. Otherwise, the maximum length of a string is a project setting.	Anywhere from 1 to 65535 Code Units per STRING is possible. Each string can have its own maximum length specified.

Programming

Individual Data types in Data type groups

Data type group	Description
INT (integer)	Any one of the following types: **U8, U16, U32, S8, S16, S32**
UINT (unsigned integer)	Any one of the following types: **U8, U16, U32**
NUMERICAL	Any one of the following types: **U8, U16, U32, U64, S8, S16, S32, S64, F32, F64**
ARRAY	Up to **32,767** elements of one of the following element types or type groups: **BOOL, NUMERICAL***
MAIN	Any one of the following types or type groups: **BOOL, U8, U16, U32, S8, S16, S32, COLOR, FILE, FONT, LANG, NULL, OBN, PORT, PXO, T, TL, ARRAY**
EXTENDED	Any one of the following types or type groups: **STRING, BOOL, NUMERICAL, ARRAY**

*Array of data types U64, S64, F32, and F64 are supported by the following GUIDE components: Non-volatile Memory Dynamic with Default, Non-volatile Memory Dynamic, Non-volatile Memory Dynamic Input, Simple Checkpoint, Advanced Checkpoint, Advanced Checkpoint with Namespace, Set Value, Cloud Write to Port, Cloud NV, Cloud Checkpoint, Cloud Set Value, Module Input, Module Output, Module Bus Input, Module Bus Output, Hardware Input Typed, Module Input Typed.

About Overflow Conditions

An overflow condition occurs when a mathematical calculation performed by a component causes an output value from that component to be either larger than the maximum or lower than the minimum value that is represented by the type used for that output pin.

When an overflow condition occurs, the output value is undefined, in some cases even outside the data type of the output pin.

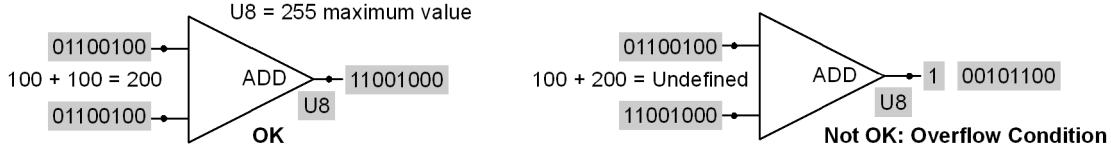
Not all components are in scope for overflow conditions. Components that are in scope, also exist in capped versions with well-defined overflow behaviors. Refer to *About Capped Components* on page 190.

Warning

Unintended movement of the machine or mechanism may cause injury to the technician or bystanders.

An overflow condition can cause unexpected, out-of-range outputs from your application. Reduce the risk of overflow conditions in your applications.

- Always consider the ranges of the data types that you use in your calculations to ensure that overflow conditions cannot occur.
- Use capped components in your calculations to enable your application to detect an overflow condition when it occurs.
- Create logic in your application to take your application to a safe state whenever it detects an overflow condition.

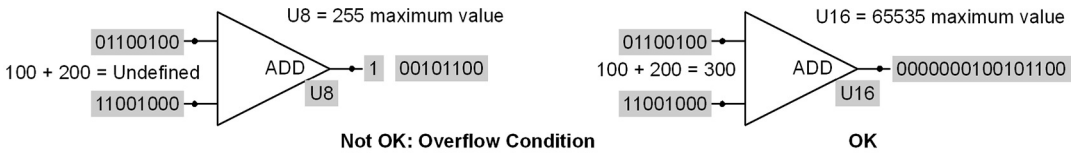


- The **Add** component adds two U8 values and outputs a U8 result.
- The **U8** data type has 8 bits, with a minimum value of 0 and a maximum value of 255:

Programming

- **OK** — 100 + 100 = 200; output is less than 255.
- **Not OK** — 100 + 200 = 300; an overflow because the output is greater than 255.

When you assign data types to integer outputs, keep in mind the minimum and maximum possible values of each data type to avoid overflow.



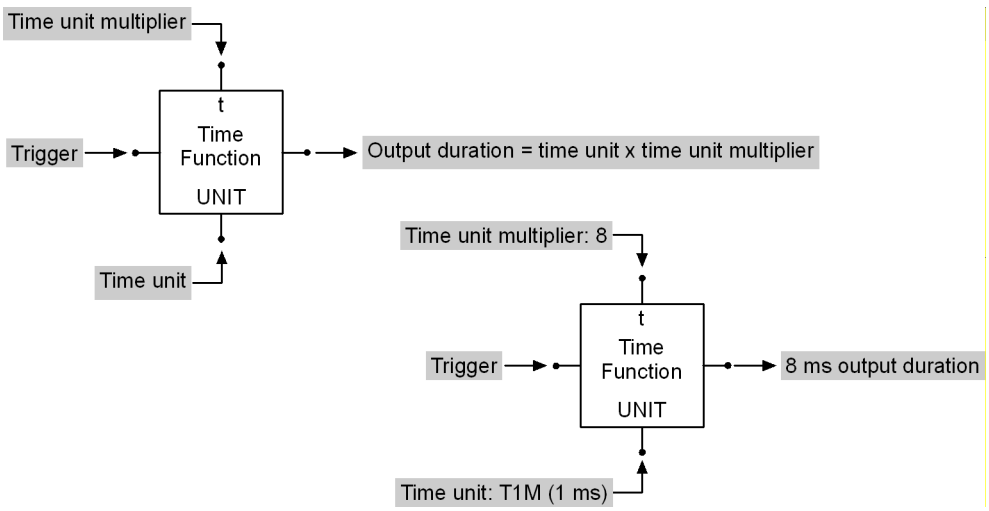
- **Not OK** — 100 + 200 = 300; an overflow condition because the output is greater than 255.
- **OK** — 100 + 200 = 300; no overflow because the U16 data type can have a maximum value of 65535.

About the Time Base data type

The Time Base data type is used with **Time and Transition** components.

Three examples of **Time and Transition** components are:

- **On Delay** — delays the return of a Boolean signal from False to True for a specified time.
- **Oscillator** — outputs a Boolean signal that oscillates at a specified frequency.
- **Pulse** — holds a Boolean output at True for a specified time before returning to False.



The preceding figure shows how time units are:

- Applied as a constant value to a time function component.
- Multiplied by the time unit multiplier to set the output duration.

The following table lists the PLUS+1 GUIDE time units and the values that they represent. Your hardware selection determines the time units that you can select.

Time Base values

Value	Abbreviation
1 ms	T1M
10 ms	T10M
100 ms	T100M
1 s	T1S

Programming

Time Base values (continued)

Value	Abbreviation
60 s	T60S
1 h	TIH
Loop time	TLOOP (Loop time or program loop—the time it takes a change in the input to produce a change in the output.)

Programming

Resolution

Processing time, output duration, and time units set the resolution (variation) between the minimum and maximum output duration times.

Minimum output duration time = output duration – time unit

Maximum output duration time = output duration + processing time

Where:

Output duration = time unit × time unit multiplier
Processing time = time for a change in an input to produce a change in the output

Example 1

Example with a time unit of 100 ms, a time unit multiplier of 5, and a processing time of 15 ms	
Min. output duration time = output duration – time unit = 400 ms	**= (100 ms × 5) – 100 ms**
Max. output duration time = output duration + processing time = 515 ms	**= (100 ms × 5) + 15 ms**

Example 2

Example with a time unit of 10 ms, a time unit multiplier of 25, and a processing time of 15 ms	
Min. output duration time = output duration – time unit = 240 ms	**= (10 ms × 25) – 10 ms**
Max. output duration time = output duration + processing time = 265 ms	**= (10 ms × 25) + 15 ms**

T Loops always have the same output duration because their time unit value is the processing time.

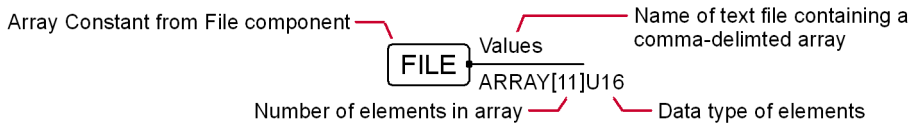
About the Array Data Type

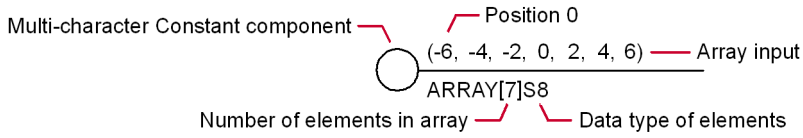
Arrays can have up to 32,767 elements. The first element in an array is in position 0.

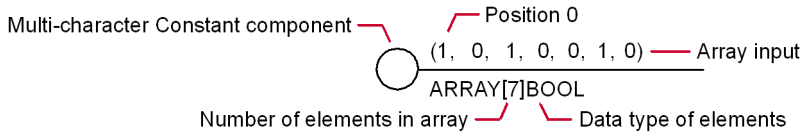
- When defining an array input from the Multi-character Constant component:
  - Enclose the array within parentheses ( ).
  - Use commas to separate each element in the array.
- When defining an array input from a text file, use commas in the text file to separate each element in the array.
- When defining the data type of an array:
  - Put the number of elements in the array within brackets [ ].
  - Select the data type of the array, such as S8, U16, or BOOL.

Programming

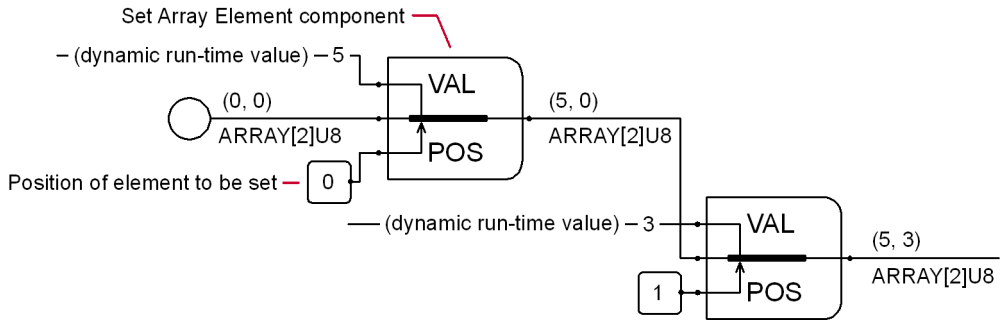
Examples of arrays and array-related components







(For Boolean arrays, 0 = F and 1 = T)



PLUS+1 GUIDE Components

Descriptions of the components under Component tab of the Selector window will be covered here.

About the Hardware - Dependency of Components

The type of hardware that you use in your project determines the components that are available for your selection from the **Component** tab.

For example, the Component tab:

- makes display components such as **Select Language, Define Areas Page,** and **Define Screen Page** available for selection if you are creating a project for graphical hardware such as the DP600.
- either hides or grays out display-related components such as **Select Language, Define Areas Page,** and **Define Screen Page** if you are creating a project for standard controller hardware such as the MC24-10 controller.

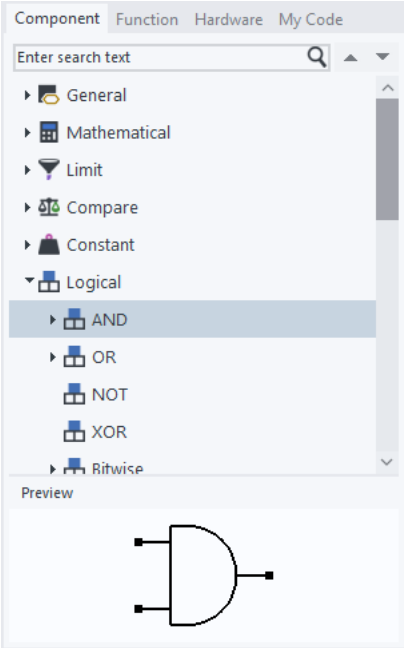
You can use:

The **Setup** menu > **Options** command > **Options** window > **Component tab hides unavailable components** setting to either hide unavailable components or display them as grayed out.

Context-sensitive Help for Components

The PLUS+1® GUIDE software has context-sensitive help for all components.

Programming



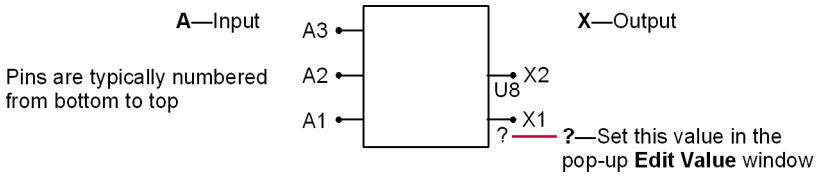
To display context-sensitive help for a component:

1. Click the **Component** for which you need a help.
2. Press **F1** button.

About Component Descriptions

This topic describes components using pictures, descriptive text, and tables.

A Typical picture of a component



In a typical component picture:

- **A** labels, such as **A1, A2,** and **A3**, indicate input pins.
- **X** labels, such as **X1** and **X2**, indicate output pins.
- **?** indicates a value that you must select if:
  - there is no data type selected.
  - you need to change the default data type.

Use either the pop-up **Edit Value** window or the **Inspector** tab to change the data type.

Programming

A typical table that lists valid connections

Valid connections for **A** input pins

Valid Connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	UINT	X2	UINT
A3	BOOL	—	——
A4	UINT	—	——

Valid connections for **X** output pins

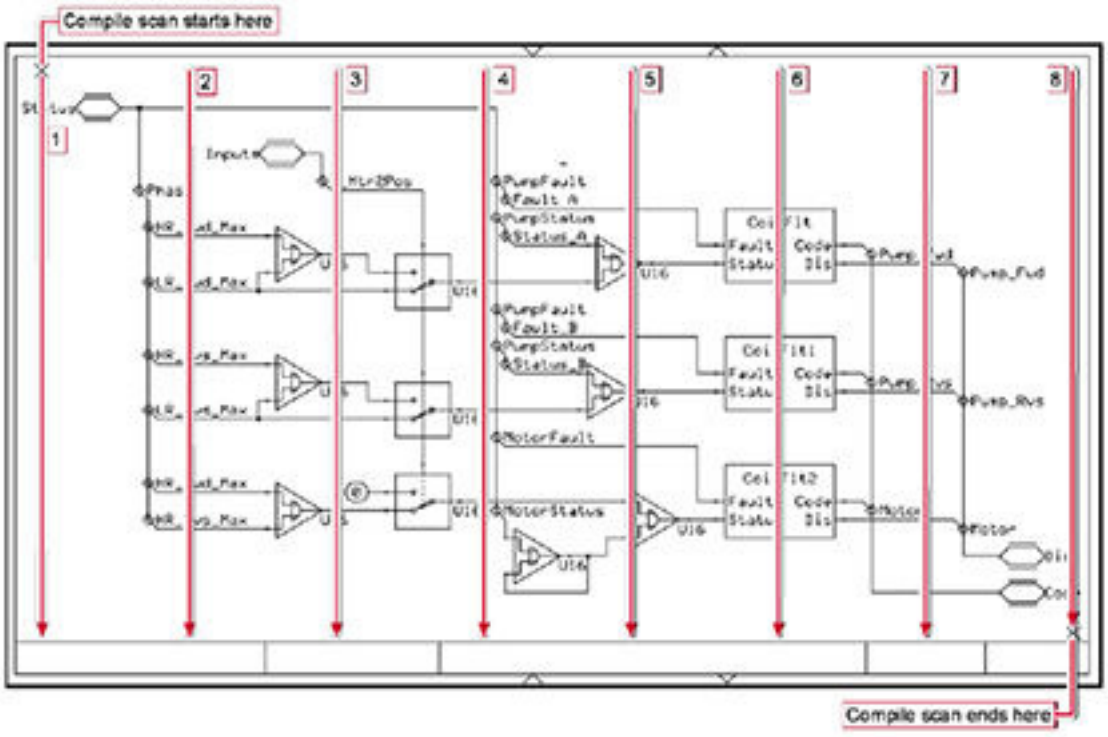
Each component has a **Valid Connections** table. This table identifies the data types that you can connect to component pins.

The table lists either single data types, such as **BOOL** or **U16**, or a data type group, such as **INT** (integer) or **UINT** (unsigned integer).

If the table lists a data type group, it means that you can connect any one of the data types in the group to a pin. For example, **INT** means that you can connect a **U8**, **U16**, **U32**, **S8**, **S16**, or **S32** data type to a pin.

See *Individual Data types in Data type groups table* under *Data Types* on page 180.

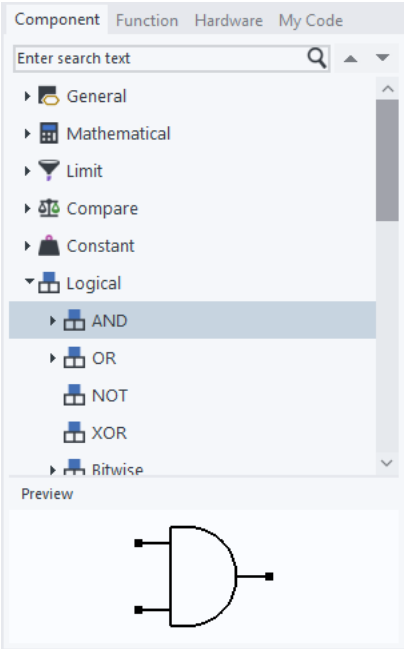
About Execution Order



Insertion points set the order in which the PLUS+1 compiler compiles code for PLUS+1 controllers. The order in which this code compiles sets the execution order of the instructions in the code.

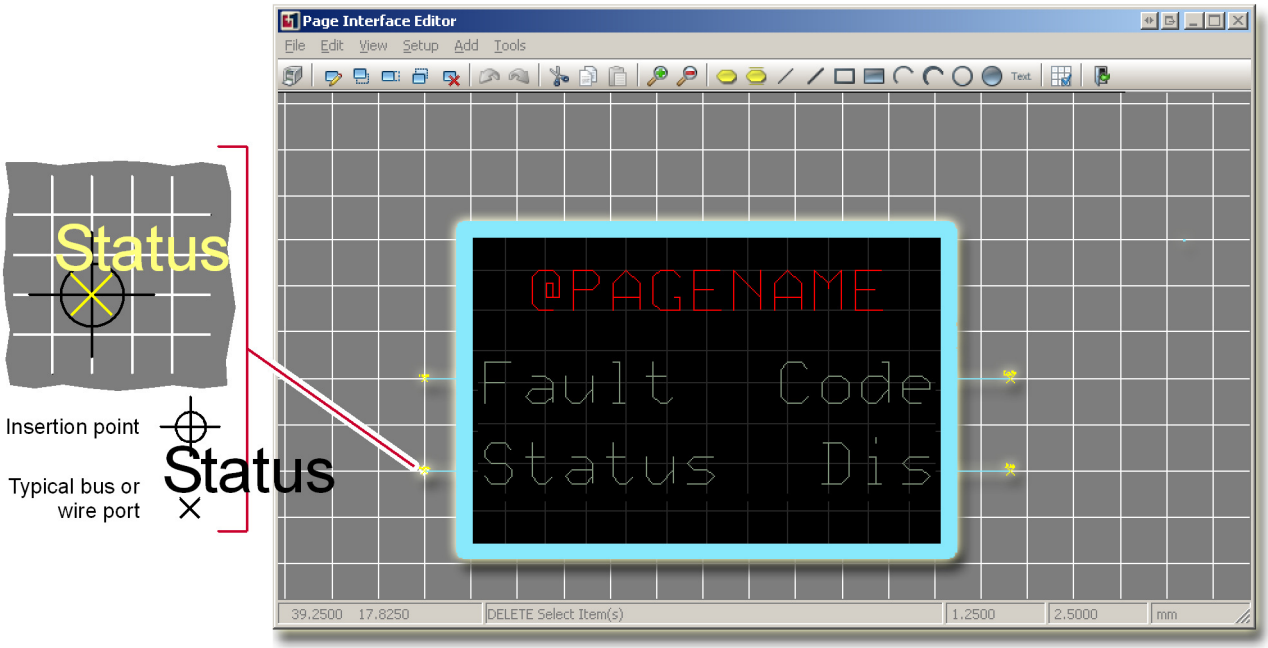
The PLUS+1 compiler compiles code for a PLUS+1 page pixel column by pixel column, starting in the page's upper-left-hand corner. The compiler moves across the page, working top-to-bottom, left-to-right.

Programming



The PLUS+1 compiler compiles code for a component starting from the component's insertion point. A component's insertion point is typically located on its lower-left pin.

The PLUS+1 cursor centers on a component's insertion point when you first drag the component from the **Component** tab.

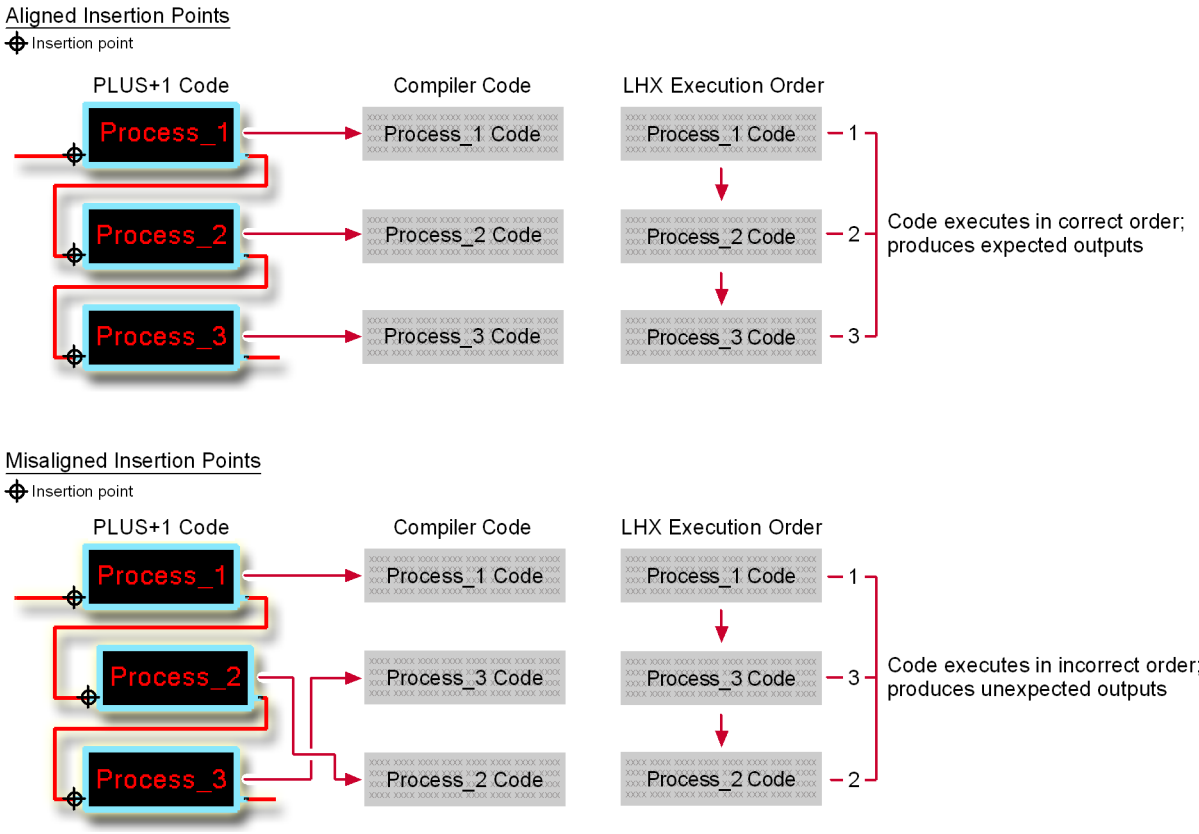


The PLUS+1 compiler compiles code for a page starting from the page's insertion point.

You set a page's insertion point in the **Page Interface Editor** window. Typically, you should align the insertion point with the page's lower-left bus port or wire port.

Programming

About misaligned Insertion Points and the execution order



Misaligned insertion points on components and on pages can produce code that compiles successfully but executes incorrectly with unexpected outputs.

The preceding figure shows how misaligned insertion points on pages can produce code that successfully compiles but executes incorrectly with unexpected outputs.

Execution order exceptions

Some components do not follow the normal rules for execution order. All components listed under the following categories follow the normal rules:

- Compare
- Logical
- Data Conversion
- Transition, Time
- Manage
- Access
- Display
- Application Log
- Page

Exceptions:

1. Initialization
   - The Capped output of the Capped components is set to False at the start of each program loop.
   - Value Initialization will set the connected net value at the start of each program loop.
2. Constants / Read Only Parameters

- These Components can be considered to be "executed" only once, before the first program loop. The associated net values will never change again after that.

3. Component output set by kernel.
- Output nets on the following components are set by the kernel between program loops:
  a. Array constant from binary file
  b. Read Array from Application Log
  c. Write Array to File
  d. Set Value v. Set Pulse
  e. Hardware sub-category
  f. CAN sub-category
  g. Non-volatile Memory Dynamic sub-category

4. Module outputs
- All module outputs are set in the same instant as their corresponding inputs are set.

5. Optimization
- When the #Unbuffered parameter is set to True, the **Call Method of Externally Defined Class** Component could modify input and output nets of array type at any time if the CCP code is not written per recommendations.

6. CAN
- CAN TX continues to transmit even from within a Module that is not called if the Send input was True when the Module was last called.
- CAN RX continues to receive even from within a Module that is not called if the CAN message matches.

About Capped Components

Capped components commonly perform a mathematical calculation. The resulting value can end up being either too large or too small to fit in the data type used to output the value from the component.

- If the value is too large to fit in the used data type defined range, then a capped component will insure the resulting value will be set to the largest value that can be represented by the used type.
- If the value is too small to fit in the data type range, then a capped component will insure the resulting value will be set to the smallest value that can be represented by the used type.

Additionally, capped components have a second output of Boolean data type. This output will be set to true whenever the calculated value does not fit in the data type range used to output the value from the component.

This second output is of connect-net type, which means it can be wired together with other outputs that are also connect-nets. This forms an OR logic functionality which can be used to see if any of a group of capped components has had to limit its calculated output value.

The only drawback with capped components, is the usage of slightly more processing and memory resources than uncapped components.

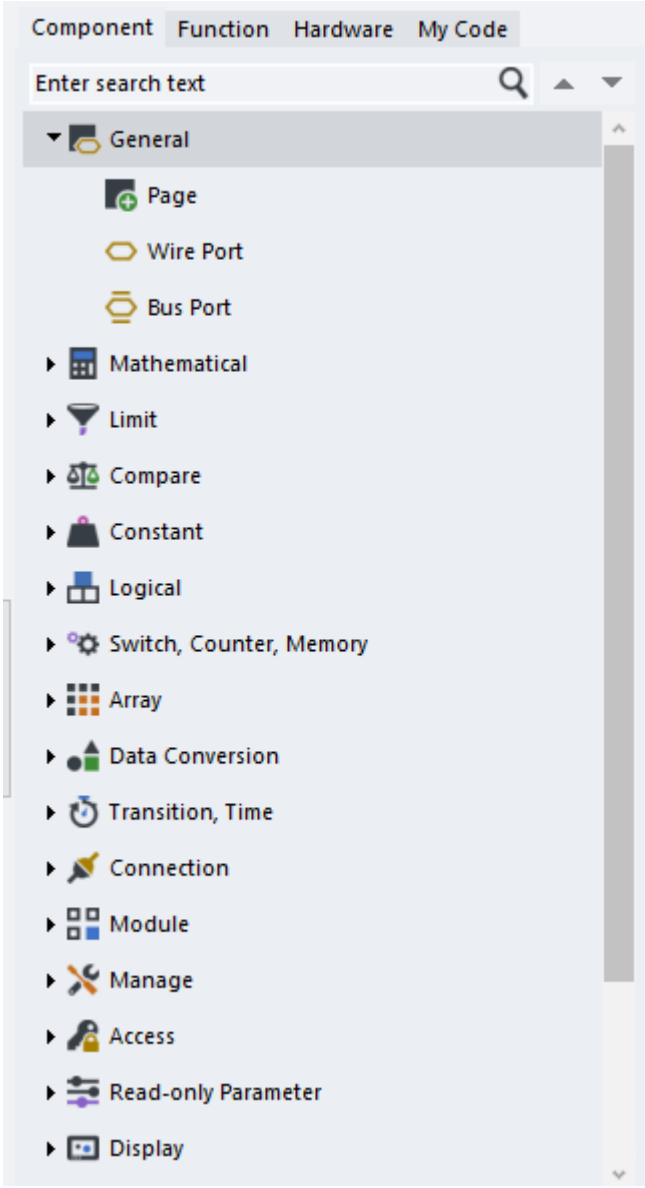
In contrast, an ordinary non-capped component performing a calculation where the resulting value does not fit in the output type, the output value will be undefined. It is the responsibility of the programmer to prevent this from happening, or handled by other code.

General recommendations:

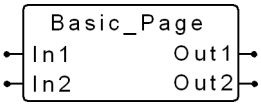
- When the occurrence of an overflow condition is uncertain, then use the capped version.
- When using the capped version of a component, always check the value of the Boolean overflow output pin.
- When there is certainty an overflow condition will not occur, then the uncapped version of a component is a good choice.

Programming

General Menu

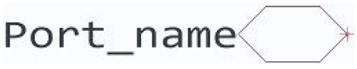


Page



Use: A generic page—use as a starting point when creating your own page.

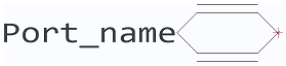
Wire Port



Use: Use this window to add a Wire Port to where you click in the Drawing Area.

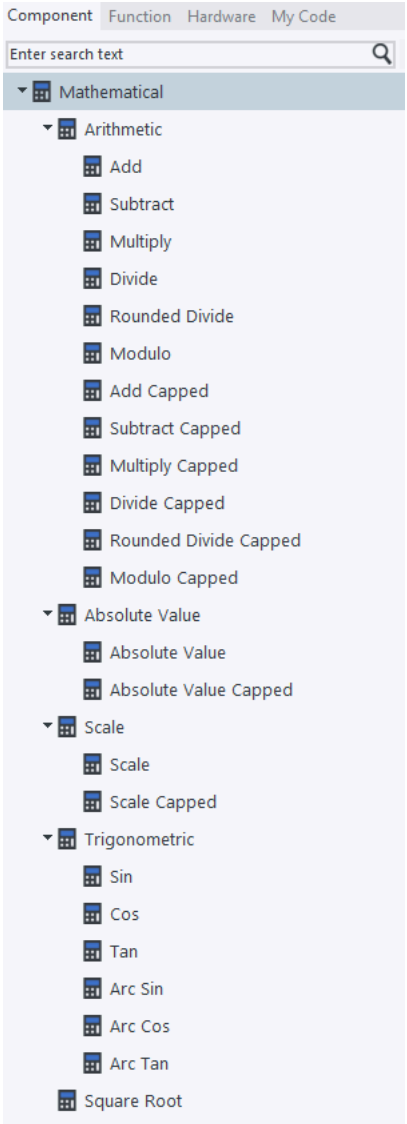
Programming

Bus Port



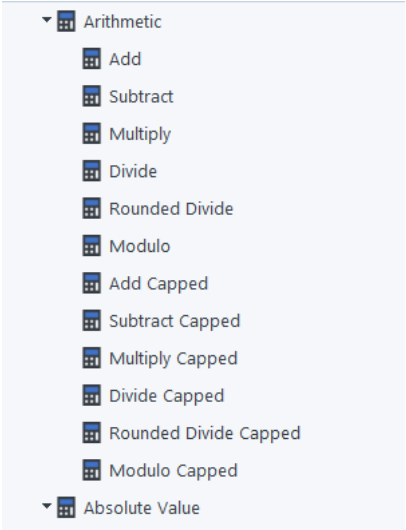
Use: Use this window to add a Bus Port to where you click in the Drawing Area.

Mathematical Menu



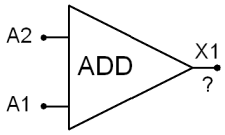
Programming

Arithmetic Menu



Add

Add symbol



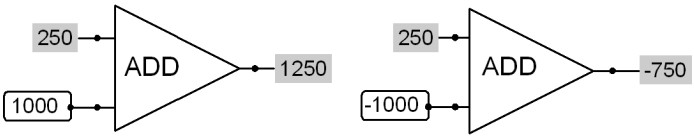
Use: Adds two integer signals.

Function: X1 = A1 + A2

Valid connections

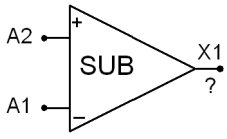
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	—	——

Example — Add



Subtract

Subtract symbol



Use: Subtracts two integer signals.

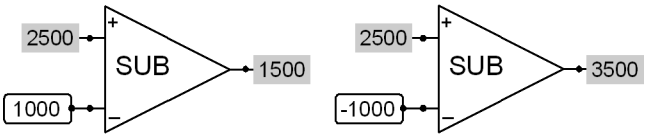
Function: X1 = A2 – A1

Programming

Valid connections

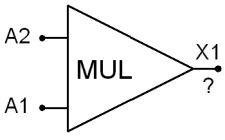
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	—	——

Example — Subtract



Multiply

Multiply symbol



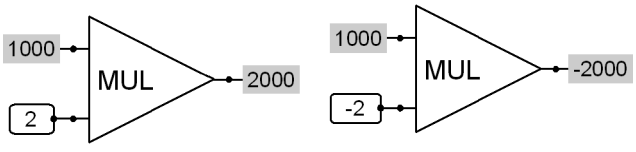
Use: Adds two integer signals.

Function: X1 = A1 ∗ A2

Valid connections

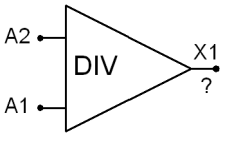
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	—	——

Example — Multiply



Divide

Divide symbol



Use: Divides two integer signals.

Function: X1 = A2 ÷ A1

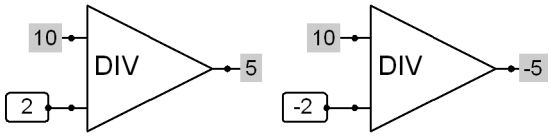
Division by Zero results in an Output value of Zero (X1 = 0 when A1 = 0)

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	—	——

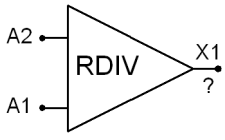
Programming

Example — Divide



Rounded Divide

Rounded Divide symbol



Use: Divides two integer signals and outputs the rounded result.

Function: X1 = Rounds the output of A2 ÷ A1

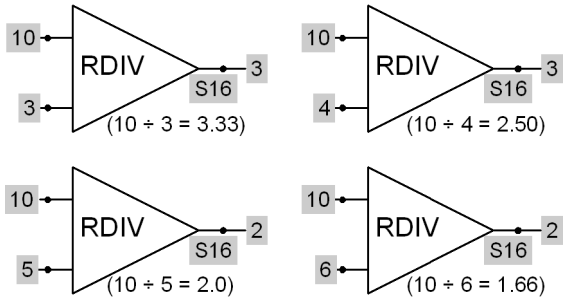
- Up if the remainder is ≥ 0.5
- Down if the remainder is < 0.5

Division by Zero results in an Output value of Zero (X1 = 0 when A1 = 0)

Valid connections

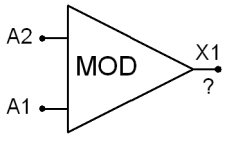
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	—	——

Example — Rounded Divide



Modulo

Modulo symbol



Use: Outputs the modulo (remainder) that results from the division of two integer signals.

Function: X1 = A2 – ([A2 /A1] ∗ A1)

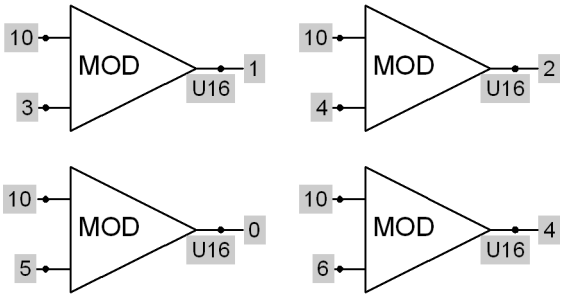
Division by Zero results in an Output value of Zero (X1 = 0 when A1 = 0)

Programming

Valid connections

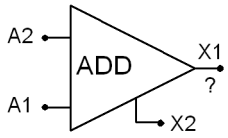
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	—	——

Example — Modulo



Add Capped

Add Capped symbol



Use:

- Adds two integer signals
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function: X1 = A1 + A2

- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

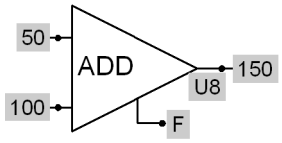
An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

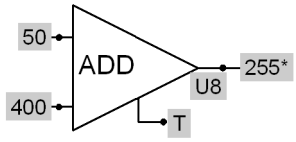
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	X2	BOOL

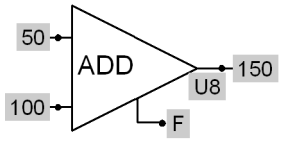
Programming

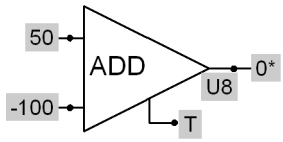
Example — Add Capped





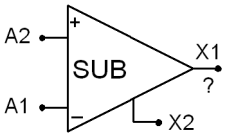
*255—Max value for the U8 data type





*0—Min value for the U8 data type

Subtract Capped



Use:

- Subtracts two integer signals
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function: X1 = A2 – A1

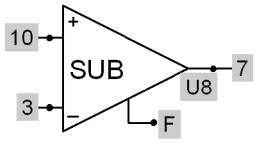
- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

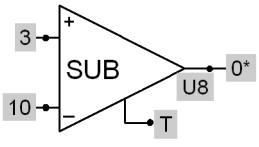
An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	X2	BOOL

Example — Subtract Capped



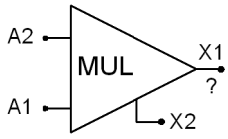


*0—Min value for the U8 data type

Programming

Multiply Capped

Multiply Capped symbol



Use:

- Multiplies two integer signals
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function: X1 = A2 ∗ A1

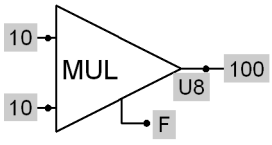
- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

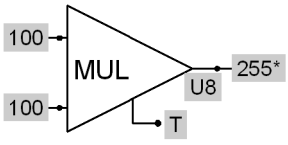
An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

Valid connections

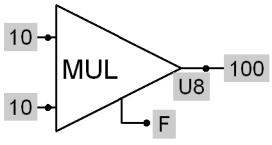
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	X2	BOOL

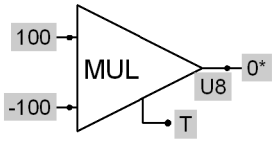
Example — Multiply Capped





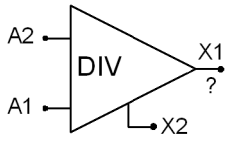
*255—Max value for the U8 data type





*0—Min value for the U8 data type

Divide Capped



Programming

Use:
- Divides two integer signals
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function: X1 = A2 ÷ A1
- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

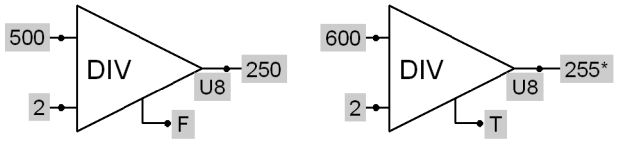
Division by Zero results in an Output value of Zero (X1 = 0 when A1 = 0)

An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	X2	BOOL

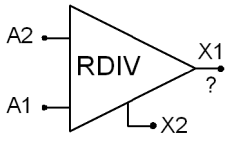
Example — Divide Capped



*255—Max value for the U8 data type

Rounded Divide Capped

Rounded Divide Capped symbol



Use:
- Divides two integer signals and outputs the rounded result
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function: X1 = Rounds the output of A2 ÷ A1:
- Up if the remainder is ≥ 0.5
- Down if the remainder is < 0.5

- If X1 does not overflow, then X2 = False
- If X1 overflows, then:

Programming

- X2 = True
- X1 = Clamps at the minimum or maximum value of its data type

- X2 resets to False at the start of each program loop

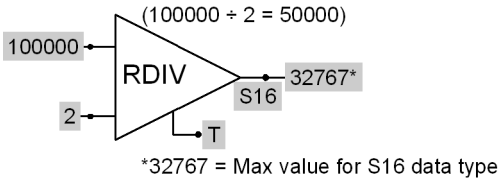
Division by Zero results in an Output value of Zero (X1 = 0 when A1 = 0)

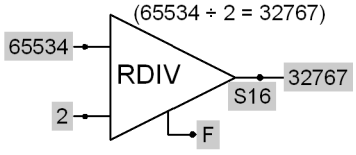
An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

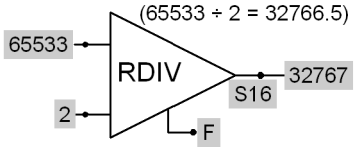
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	X2	BOOL

Example — Rounded Divide Capped

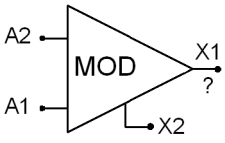






Modulo Capped

Modulo Capped symbol



Use:

- Outputs the modulo (remainder) that results from the division of two integer signals
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function: X1 = A2 – ([A2 /A1] ∗ A1)

- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

Division by Zero results in an Output value of Zero (X1 = 0 when A1 = 0)

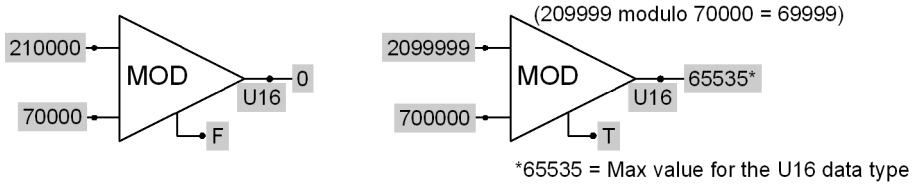
An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

Programming

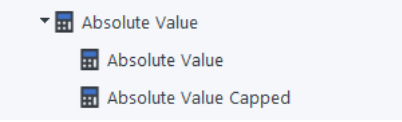
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	X2	BOOL

Example — Modulo Capped

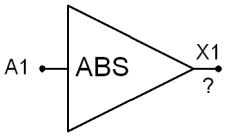


Absolute Value Menu



Absolute Value

Absolute Value symbol



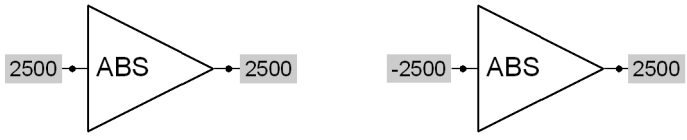
Use: Outputs the magnitude of an integer signal.

Function: X1 = Absolute value of A1

Valid connections

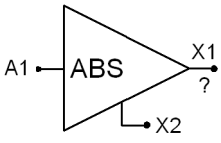
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT

Example — Absolute Value



Absolute Value Capped

Absolute Value Capped symbol



Programming

Use:
- Outputs the magnitude of an integer signal
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function: X1 = Absolute value of A1:
- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

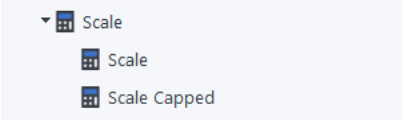
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
—	——	X2	BOOL

Example — Absolute Value Capped

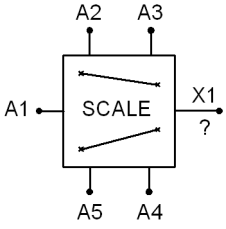
*255—Max value for the U8 data type

Scale Menu



Scale

Scale symbol



Use:
- Scales an integer signal into a new range
- Applies scaling to a sensor signal

Function:

$$X1 = \left(\left(\frac{A1 - A5}{A2 - A5}\right) \times (A3 - A4)\right) + A4$$

Programming

The A2–A5 values set the input-to-output ratio and offset applied to the A1 input signal. They do not limit the resulting X1 output value.

When A2 = A5, the output from the Scale component is undefined.

The result is rounded according to the following rules:

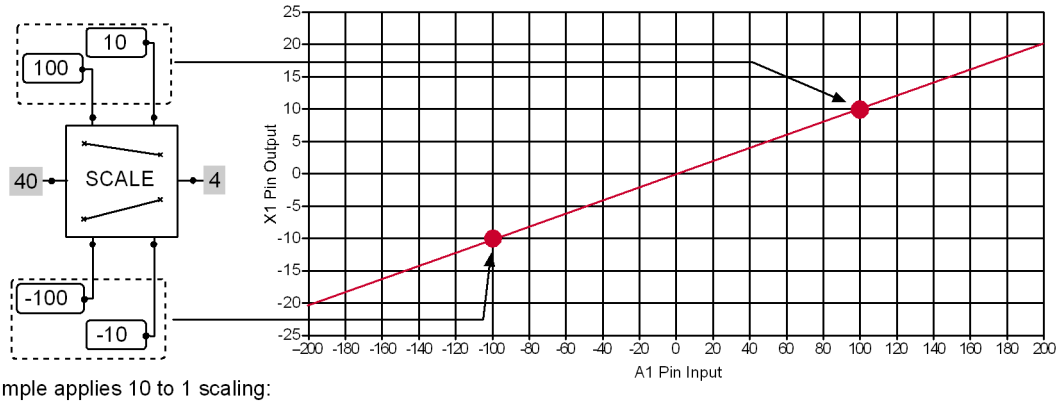
- Positive values are rounded up if the fractional part is ≥ 0.5 and down otherwise.
- Negative values are rounded down if the fractional part of the absolute value is ≥ 0.5 and up otherwise.

Valid connections

Pin	Data Type	Pin	Data Type
A1–A5	INT	X1	INT

The following two figures and plots show how different input values applied to this component produce different input/output ratios. This example applies to both Scale components.

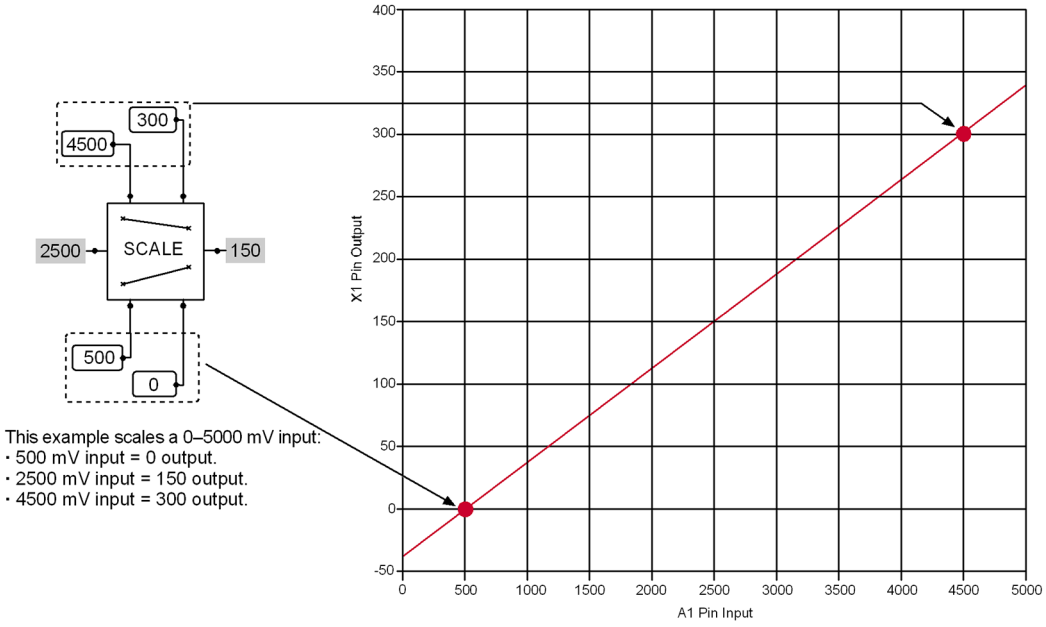
Example 1 — Scale



This example applies 10 to 1 scaling:
- -100 input = -10 output.
- 40 input = 4 output.
- 100 input = 10 output.

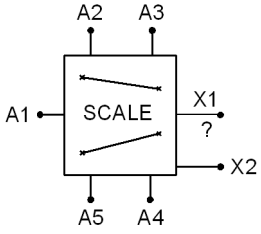
Programming

Example 2 — Scale



Scale Capped

Scale Capped symbol



Use:

- Scales an integer signal into a new range
- Applies scaling to a sensor signal
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function: $X1 = \left(\left(\frac{A1 - A5}{A2 - A5}\right) \times (A3 - A4)\right) + A4$

- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

This component has an unpredictable X2 (True = Overflow) output when used in pages whose **Object** property is **True**. However, its X1 output works correctly on these pages.

The A2–A5 values set the input-to-output ratio and offset applied to the A1 input signal. They do not limit the resulting X1 output value.

Programming

An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

When A2 = A5, the output from the Scale Capped component is undefined.

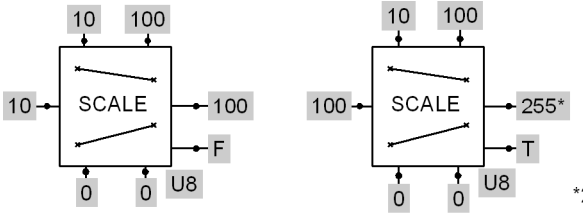
The result is rounded according to the following rules:

- Positive values are rounded up if the fractional part is ≥ 0.5 and down otherwise.
- Negative values are rounded down if the fractional part of the absolute value is ≥ 0.5 and up otherwise.

Valid connections

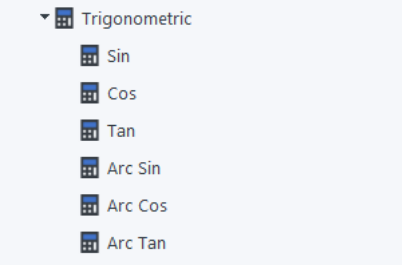
Pin	Data Type	Pin	Data Type
A1–A5	INT	X1	INT
—	——	X2	BOOL

Example — Scale Capped

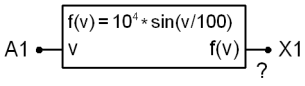


*255—Max value for the U8 data type

Trigonometric Menu



Sin



Use: Outputs the sine function of an integer signal.

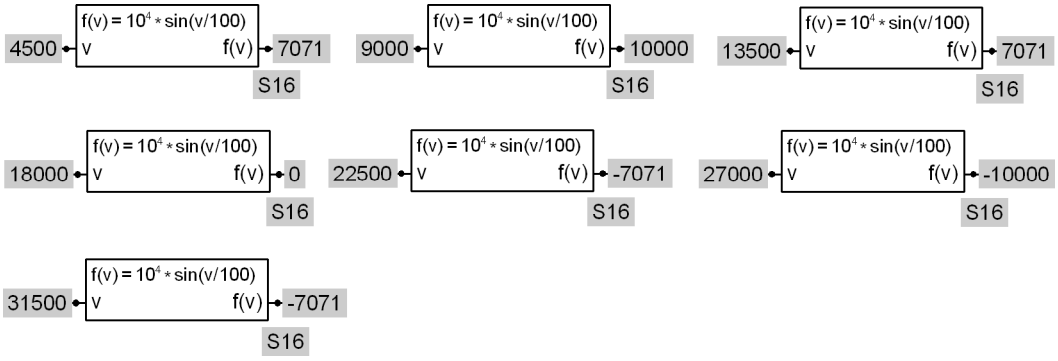
Function: X1 = 10^4 x sin (A1/100)

Valid connections

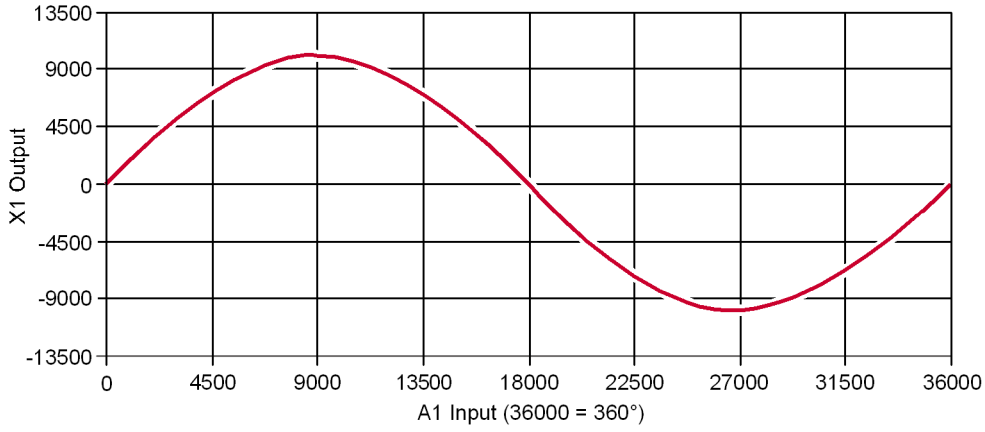
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT

Programming

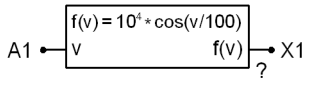
Example — Sin



X1 Output vs. A1 Input



Cos



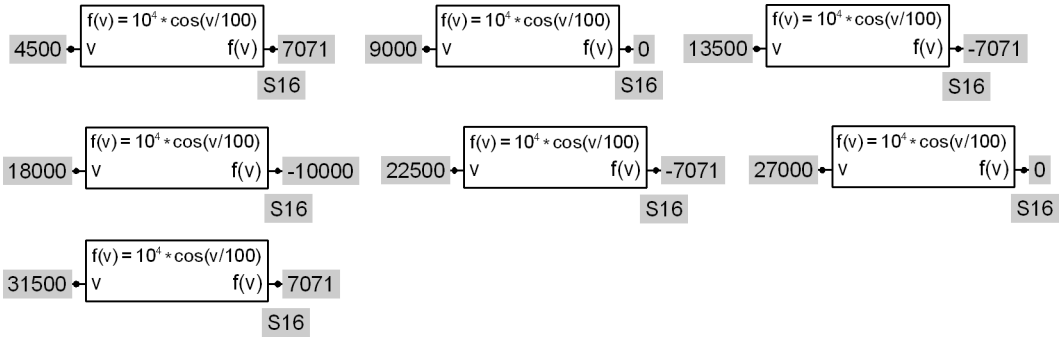
Use: Outputs the cosine function of an integer signal.

Function: X1 = 10^4 x cos (A1/100)

Valid connections

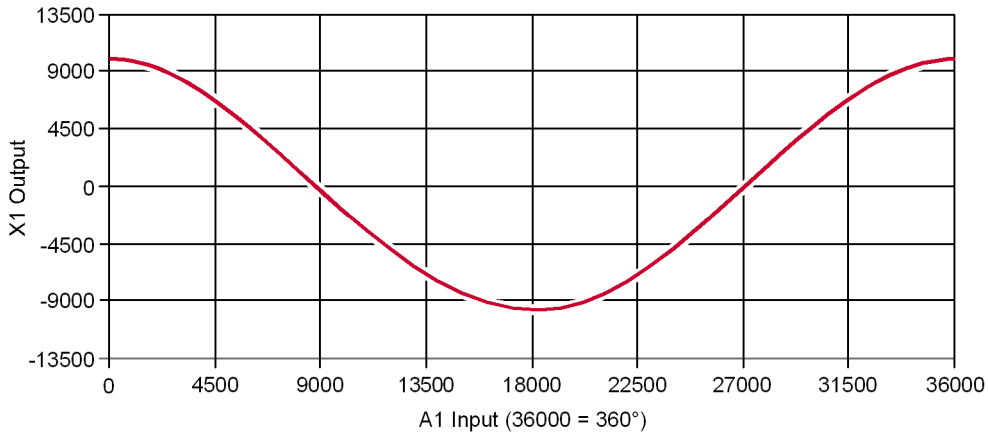
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT

Example — Cos

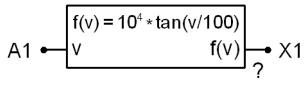


Programming

X1 Output vs. A1 Input



Tan



Use: Outputs the tangent function of an integer signal.

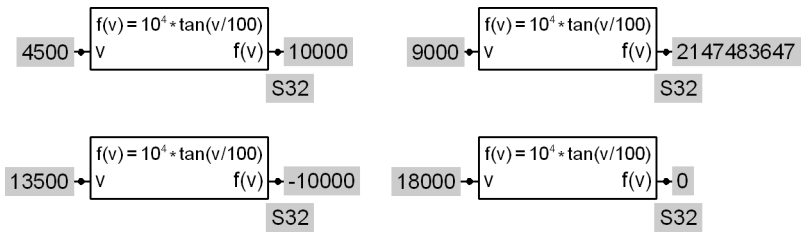
Tan(x) is a discontinuous function. This component will always output 2147483647 in place of an undefined value. This applies to V = 9000 (=90 degrees) plus or minus any multiple of 180 degrees. (Meaning that for example v = -9000 will also yield an output value of 2147483647.)

Function: X1 = 10^4 x tan (A1/100)

Valid connections

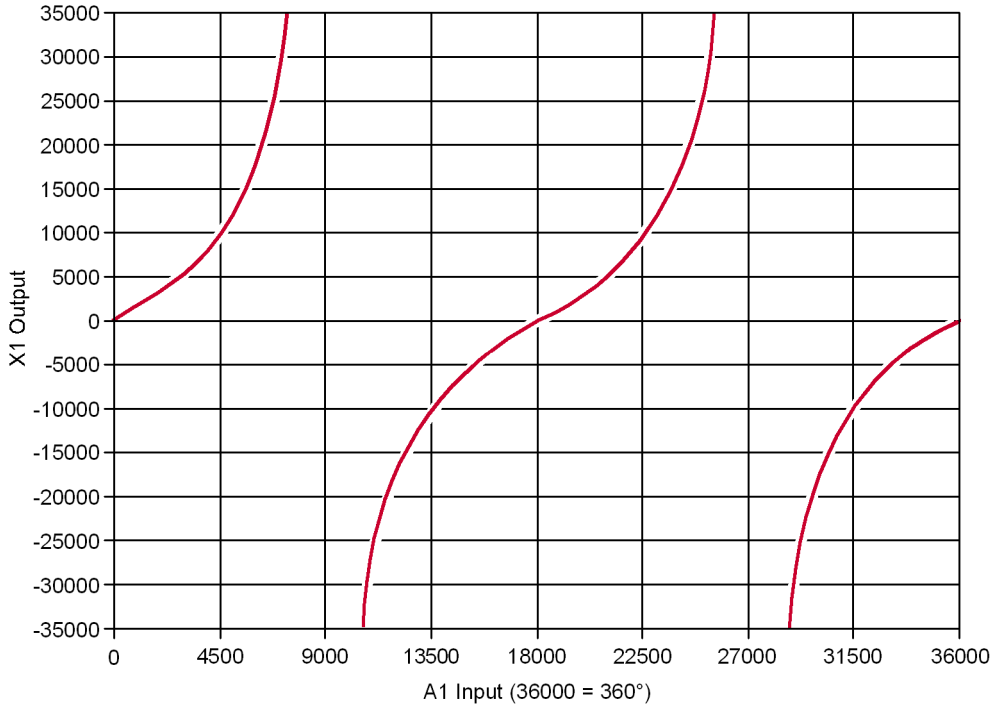
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT

Example — Tan

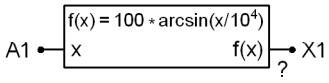


Programming

X1 Output vs. A1 Input



Arc Sin



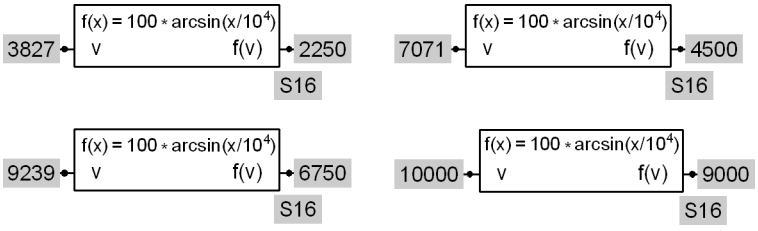
Use: Outputs the arcsine function of an integer signal.

Function: X1 = 100 x arcsin (A1/10^4)

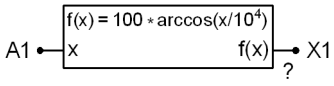
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT

Example — Arc Sin



Arc Cos



Use: Outputs the arccosine function of an integer signal.

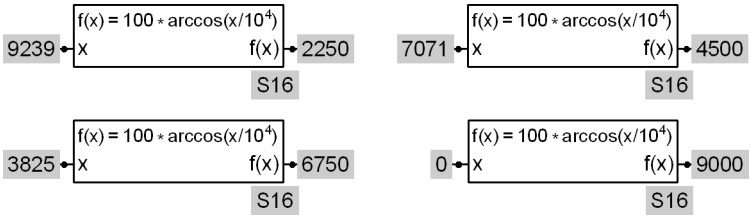
Function: X1 = 100 x arccos (A1/10^4)

Programming

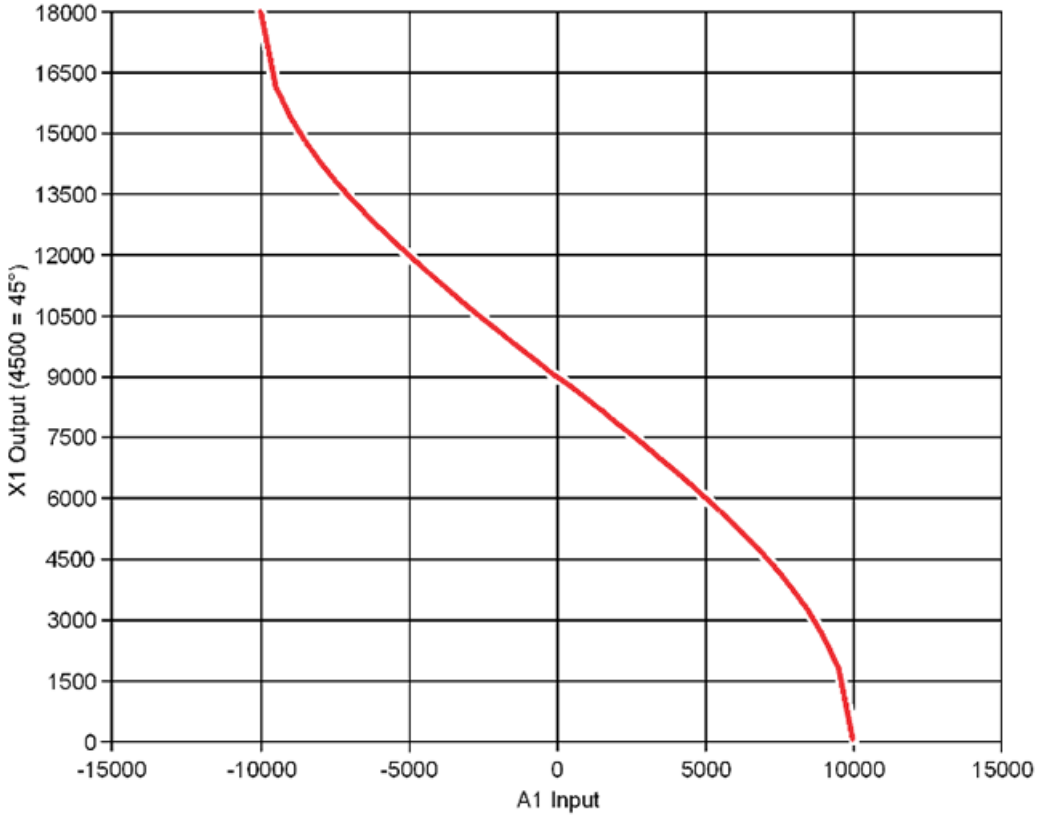
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT

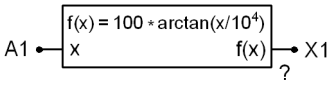
Example — Arc Cos



X1 Output vs. A1 Input



Arc Tan



Use: Outputs the arctangent function of an integer signal.

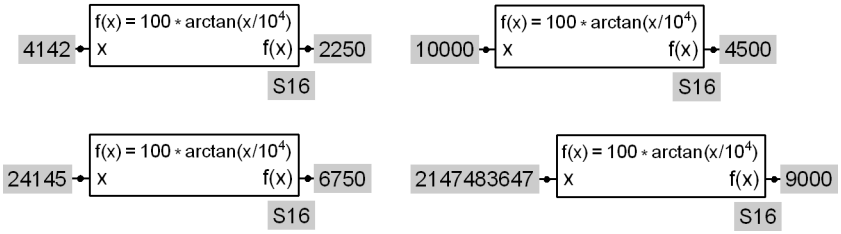
Function: X1 = 100 x arctan (A1/10^4)

Valid connections

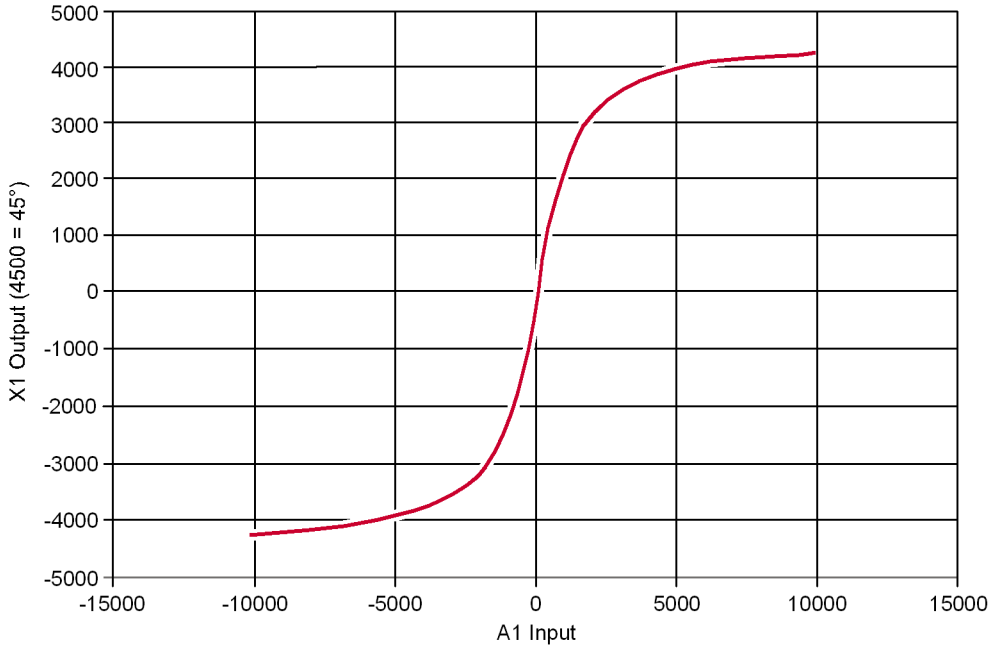
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT

Programming

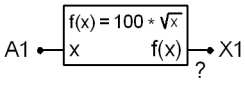
Example — Arc Tan



X1 Output vs. A1 Input



Square Root



Use: Outputs the square root of an integer signal.

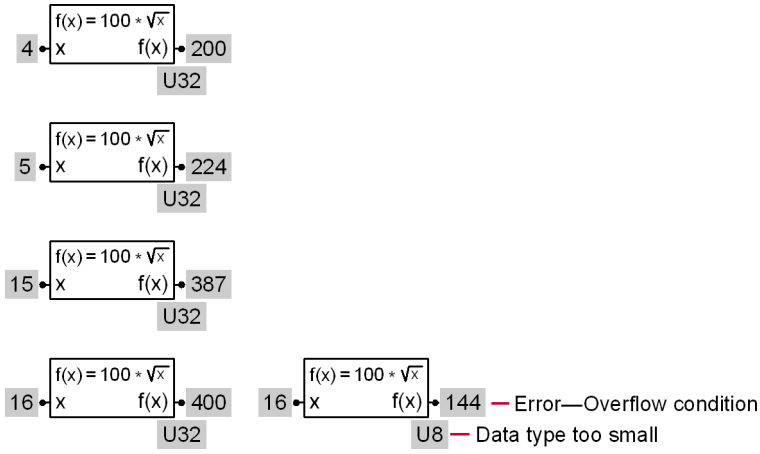
Function: $X1 = 100 \times \sqrt{A1}$ {Accuracy ±4}

Valid connections

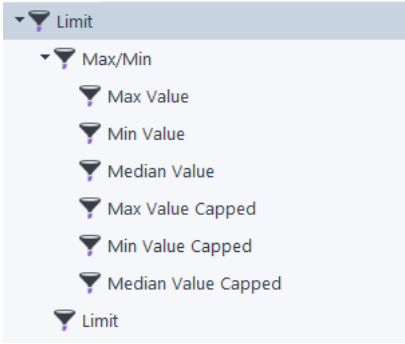
Pin	Data Type	Pin	Data Type
A1	UINT	X1	UINT

Programming

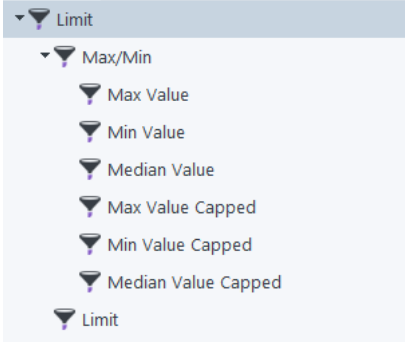
Example — Square Root



Limit Menu

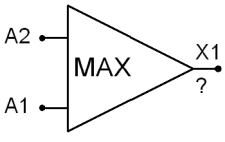


Max/Min Menu



Max Value

Max Value symbol



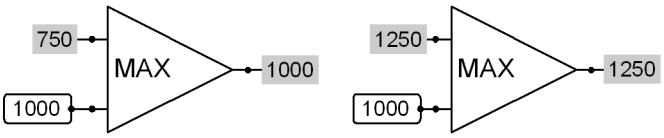
Use:	Outputs the larger value of two integer signal.
Function:	X1 = A1 if A1 ≥ A2
	X1 = A2 if A2 > A1

Programming

Valid connections

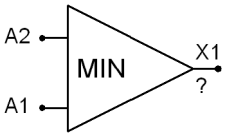
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	—	——

Example — Max Value



Min Value

Max Value symbol



Use: Outputs the smaller value of two integer signal.

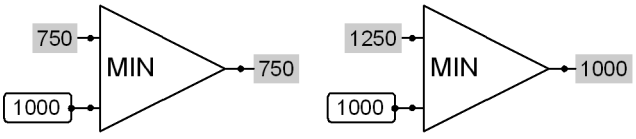
Function: X1 = A1 if A1 ≤ A2

X1 = A2 if A2 < A1

Valid connections

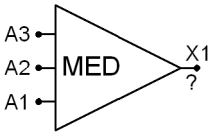
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	—	——

Example — Min Value



Median Value

Median Value symbol



Use: Outputs the median value of three input values

Set upper and lower output limits

Function: X1 = A1 if:

- A2 ≤ A1 ≤ A3
- A3 ≤ A1 ≤ A2

X1 = A2 if:

Programming

- A1 ≤ A2 ≤ A3
- A3 ≤ A2 ≤ A1

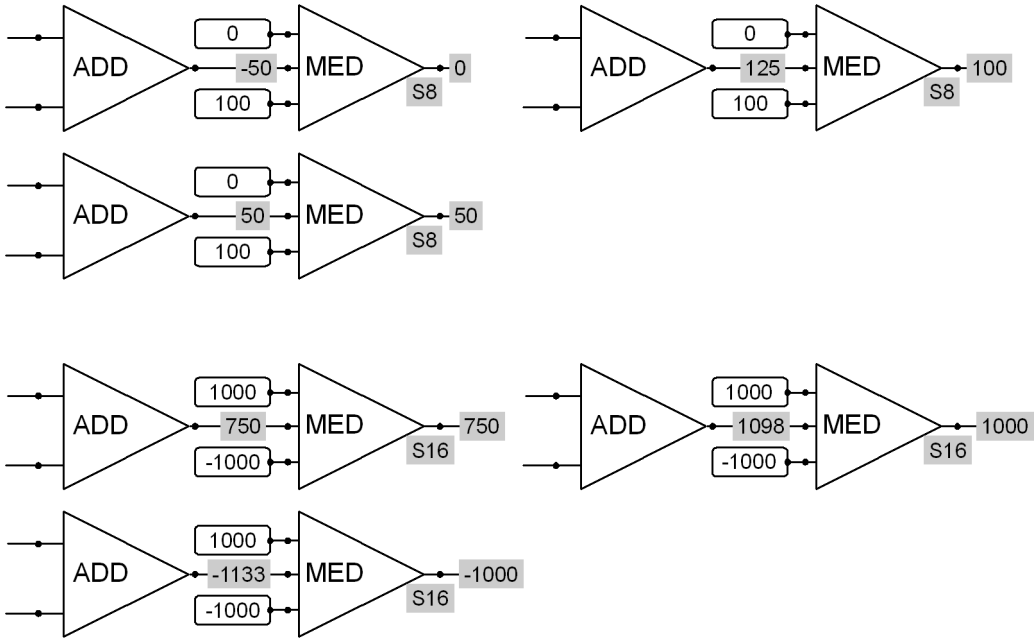
X1 = A3 if:

- A1 ≤ A3 ≤ A2
- A2 ≤ A3 ≤ A1

Valid connections

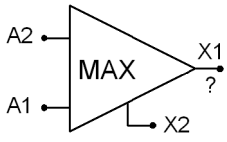
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	—	——
A3	INT	—	——

Example — Median Value



Max Value Capped

Max Value Capped symbol



Use:

- Outputs the larger value of two integer signals
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function: X1 = Rounds the output of A2 ÷ A1:

Programming

- X1 = A1 if A1 ≥ A2
- X1 = A2 if A2 > A1

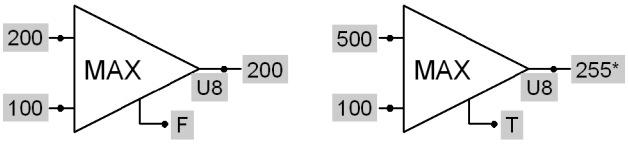
- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	X2	BOOL

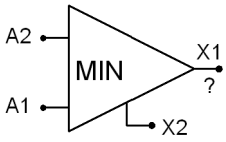
Example — Max Value Capped



*255—Max value for the U8 data type

Min Value Capped

Min Value Capped symbol



Use:

- Outputs the smaller value of two integer signals
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function: X1 = Rounds the output of A2 ÷ A1:

- X1 = A1 if A1 ≤ A2
- X1 = A2 if A2 < A1

- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

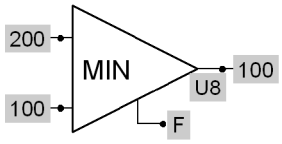
An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

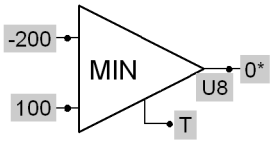
Programming

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	X2	BOOL

Example — Min Value Capped

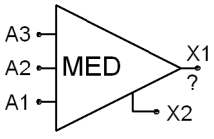




*0—Min value for the U8 data type

Median Value Capped

Median Value Capped symbol



Use:

- Outputs the median value of three integer signals
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net
- Set upper and lower output limits

Function: X1 = A1 if:

- A2 ≤ A1 ≤ A3
- A3 ≤ A1 ≤ A2

X1 = A2 if:

- A1 ≤ A2 ≤ A3
- A3 ≤ A2 ≤ A1

X1 = A3 if:

- A1 ≤ A3 ≤ A2
- A2 ≤ A3 ≤ A1

If X1 overflows, then:

- X2 = True
- X1 = Clamps at the minimum or maximum value of its data type

X2 resets to False at the start of each program loop.

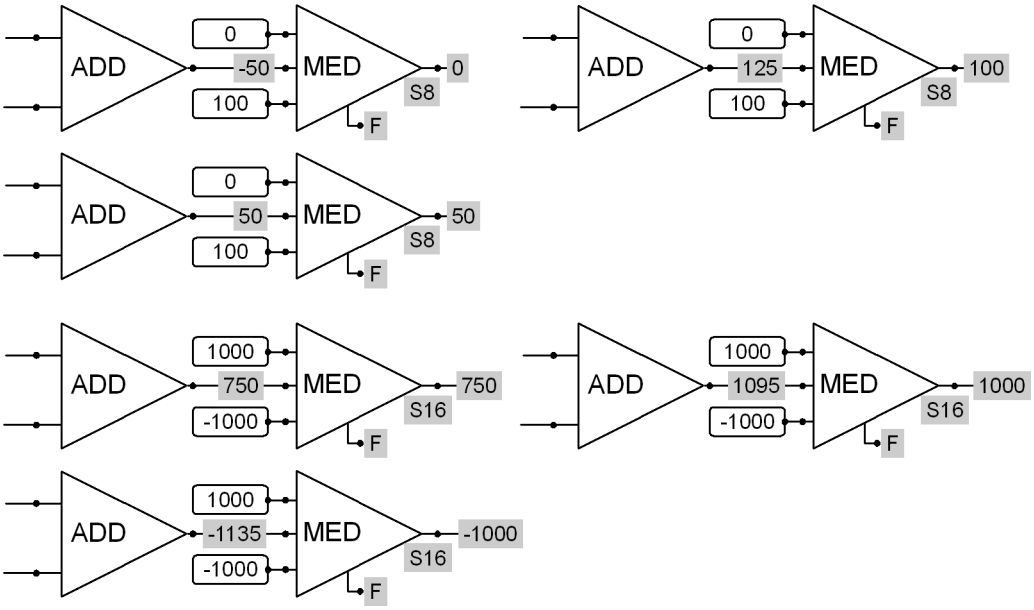
An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

Programming

Valid connections

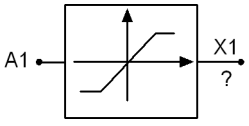
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	X2	BOOL
A3	INT	—	——

Example — Median Value Capped



Limit

Limit symbol



Use: Limits the X1 output to the maximum and minimum values of the data type selected for X1

Function: X1 = A1 converts into X1 data type

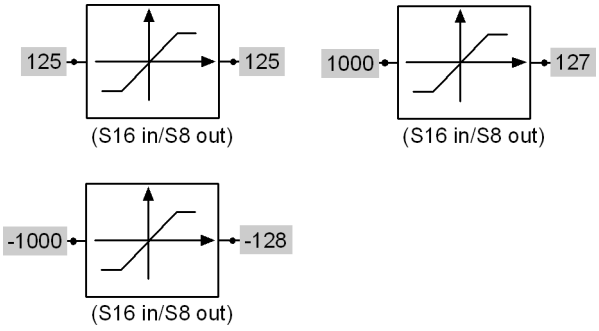
- X1 = Maximum value of the X1 data type if A1 > maximum value of the X1 data type
- X1 = Minimum value of the X1 data type if A1< minimum value of the X1 data type

Valid connections

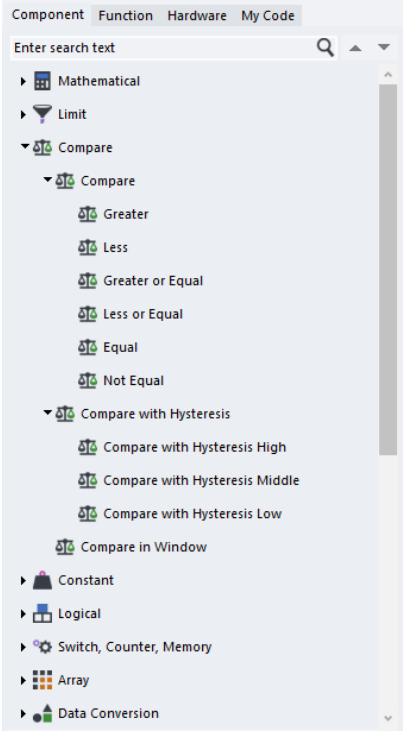
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT

Programming

Example — Limit

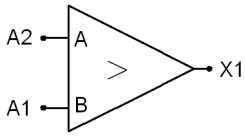


Compare Menu



Greater

Greater symbol



Use: Outputs a True if A is greater than B

Function: Compares A2 to A1

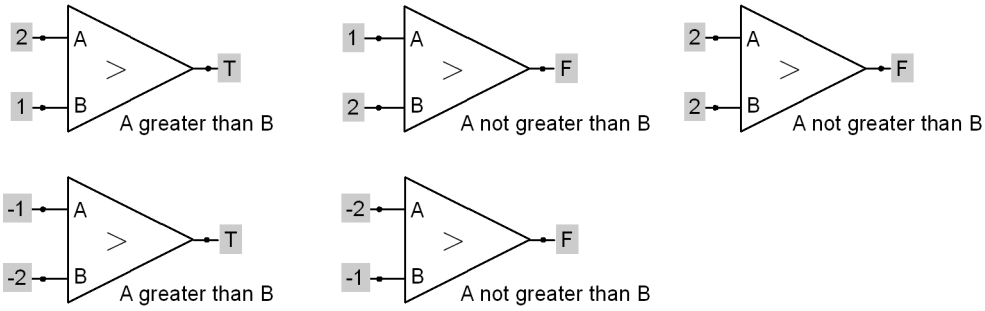
- X1 = True if A2 > A1
- X1 = False if A2 = A1
- X1 = False if A2 < A1

Programming

Valid connections

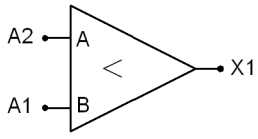
Pin	Data Type	Pin	Data Type
A1	INT	X1	BOOL
A2	INT	—	——

Example — Greater



Less

Less symbol



Use: Outputs a True if A is less than B

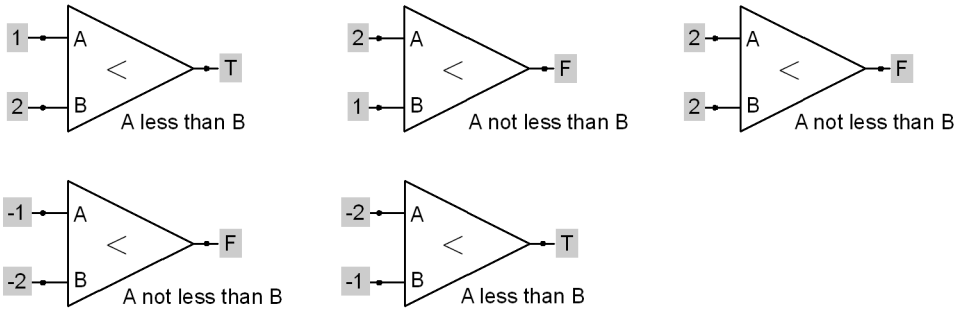
Function: Compares A2 to A1

- X1 = True if A2 < A1
- X1 = False if A2 = A1
- X1 = False if A2 > A1

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	BOOL
A2	INT	—	——

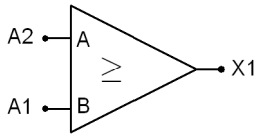
Example — Less



Programming

Greater or Equal

Greater or Equal symbol



Use: Outputs a True if A is greater than or equal to B

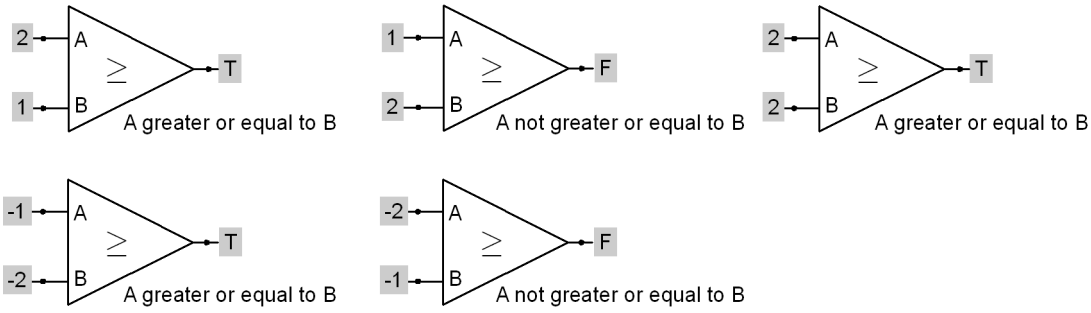
Function: Compares A2 to A1

- X1 = False if A2 < A1
- X1 = True if A2 = A1
- X1 = True if A2 > A1

Valid connections

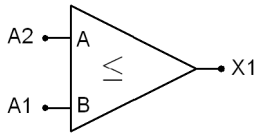
Pin	Data Type	Pin	Data Type
A1	INT	X1	BOOL
A2	INT	—	——

Example — Greater or Equal



Less or Equal

Less or Equal symbol



Use: Outputs a True if A is less than or equal to B

Function: Compares A2 to A1

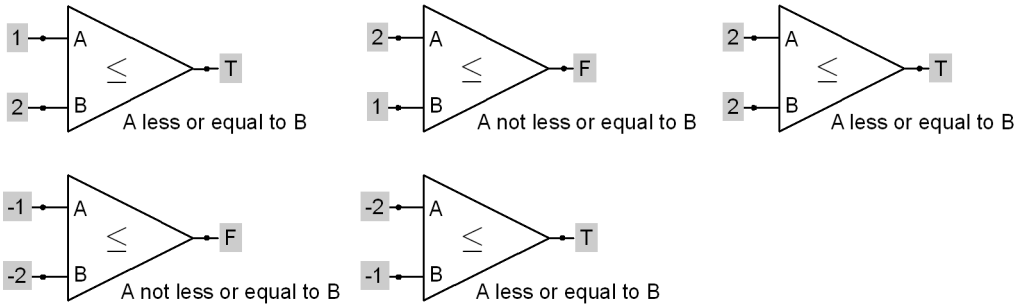
- X1 = True if A2 < A1
- X1 = True if A2 = A1
- X1 = False if A2 > A1

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	BOOL
A2	INT	—	——

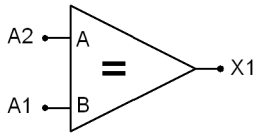
Programming

Example — Less or Equal



Equal

Equal symbol



Use: Outputs a True if A is equal to B

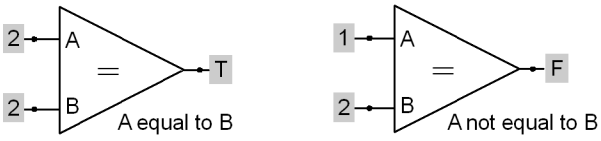
Function: Compares A2 to A1

- X1 = False if A2 < A1
- X1 = True if A2 = A1
- X1 = False if A2 > A1

Valid connections

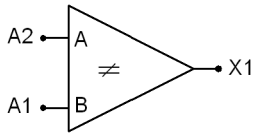
Pin	Data Type	Pin	Data Type
A1	INT	X1	BOOL
A2	INT	—	——

Example — Equal



Not Equal

Not Equal symbol



Use: Outputs a True if A is not equal to B

Function: Compares A2 to A1

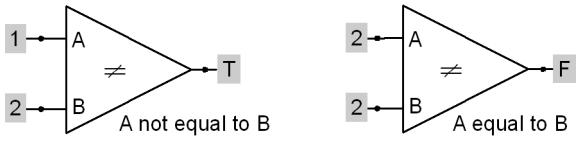
Programming

- X1 = True if A2 ≠ A1
- X1 = False if A2 = A1

Valid connections

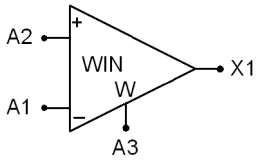
Pin	Data Type	Pin	Data Type
A1	INT	X1	BOOL
A2	INT	—	——

Example — Not Equal



Compare in Window

Compare in Window symbol



Use: Checks if the absolute difference between two signals is within a specified range and then outputs a Boolean value based on this comparison

Function:

- X1 = True if A2 ≥ A1 – (A3 ÷ 2) and A2 ≤ A1 + (A3 ÷ 2)
- X1 = False if A2 > A1 + (A3 ÷ 2)
- X1 = False if A2 < A1 – (A3 ÷ 2)

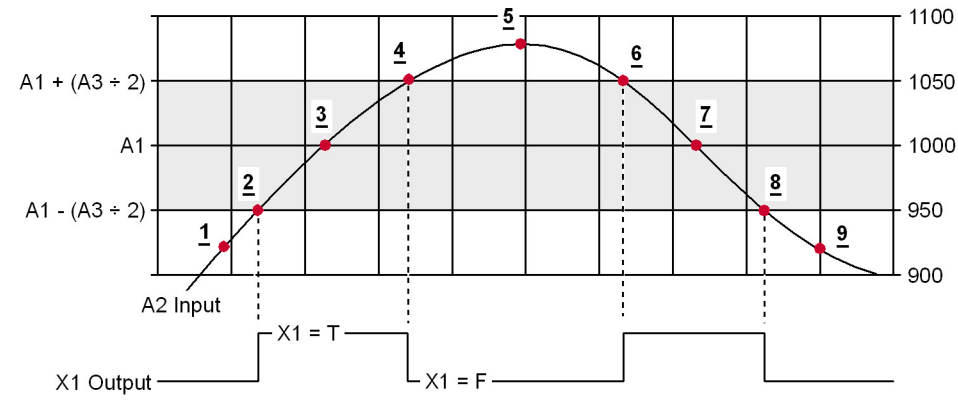
  The product of A3 ÷ 2 rounds down to the next lowest integer (for example, 7.5 becomes 7).

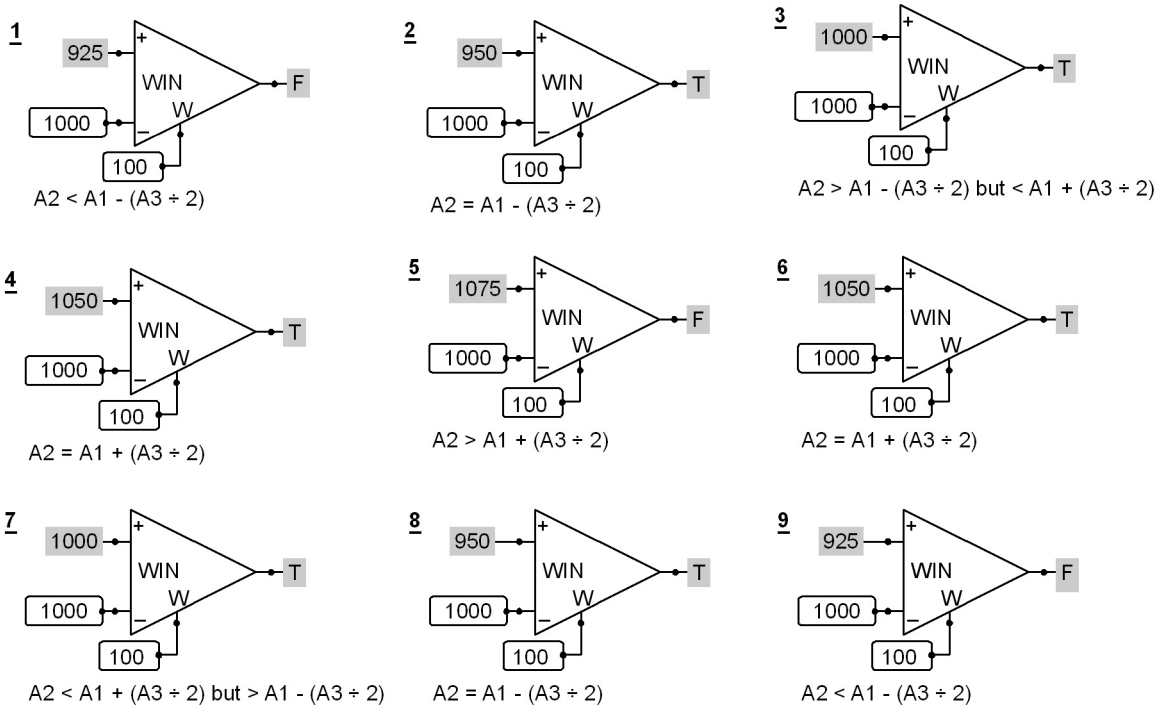
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	BOOL
A2	INT	—	——
A3	UINT	—	——

Programming

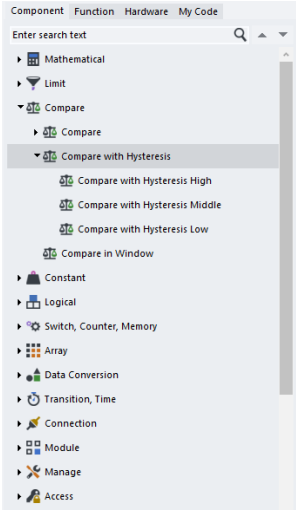
Example — Compare in Window





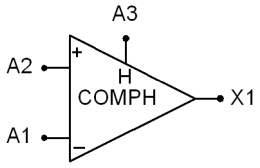
Programming

Compare with Hysteresis Menu



Compare with Hysteresis High

Compare with Hysteresis High symbol



Use: Outputs a Boolean value based on a hysteresis comparison between two integer input signals

Function:

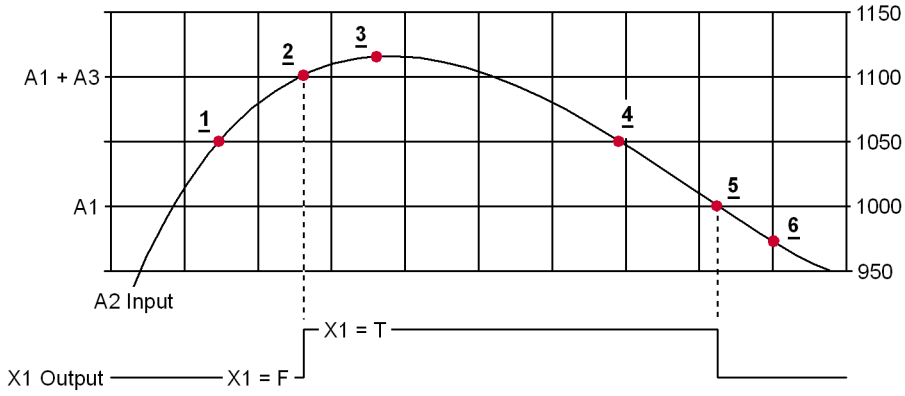
- X1 = True if A2 ≥ (A1 + A3)
- X1 = False if A2 < A1
- X1 = No change if A1 ≤ A2 < (A1 + A3)

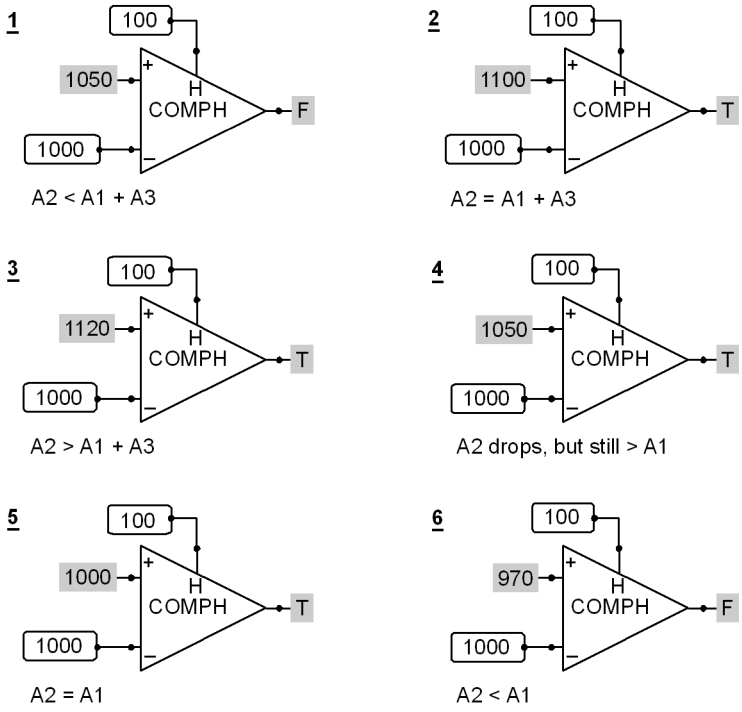
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	BOOL
A2	INT	—	——
A3	UINT	—	——

Programming

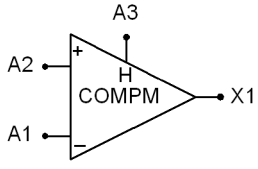
Example — Compare with Hysteresis High





Compare with Hysteresis Middle

Compare with Hysteresis Middle symbol



Use: Outputs a Boolean value based on a hysteresis comparison between two integer input signals

Function:

Programming

- X1 = True if A2 ≥ A1 + (A3 ÷ 2)
- X1 = False if A2 < A1 – (A3 ÷ 2)
- X1 = No change if A1 – (A3 ÷ 2) ≤ A2 < A1 + (A3 ÷ 2)

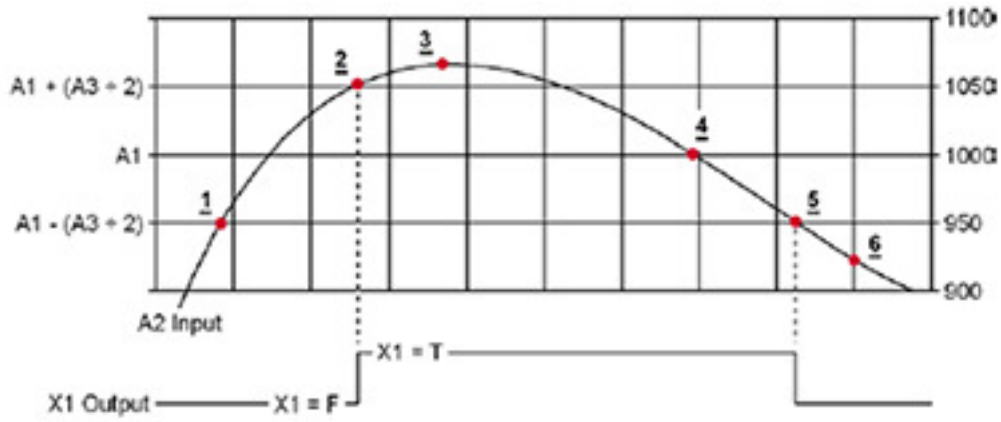
  The quotient of A3 ÷ 2 rounds down to the next lowest integer (for example, 7.5 becomes 7).

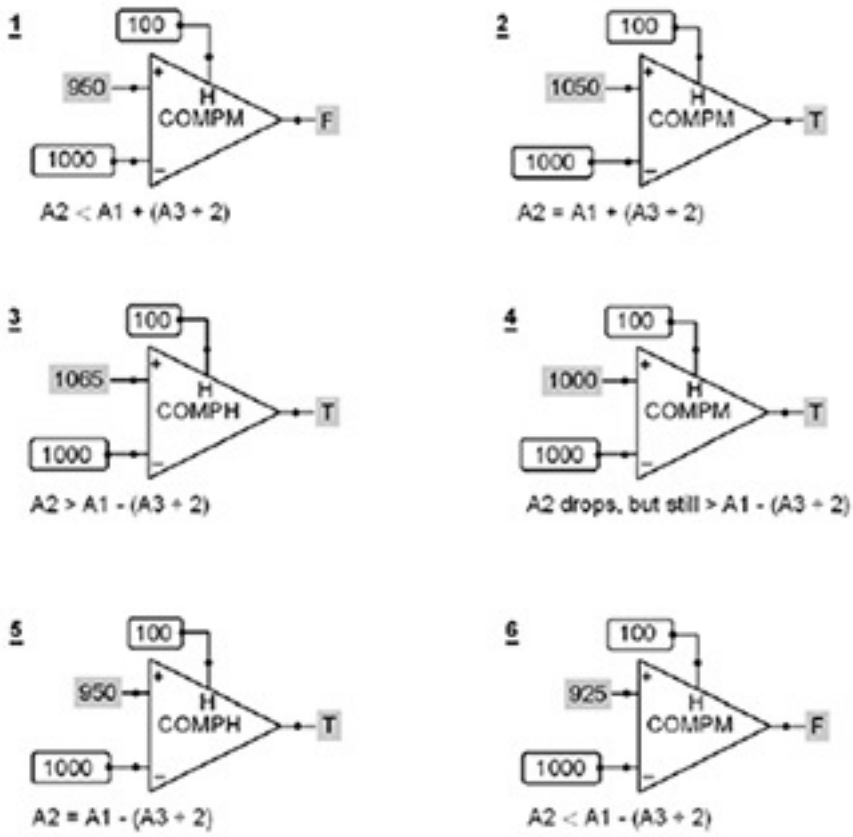
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	BOOL
A2	INT	—	——
A3	UINT	—	——

Programming

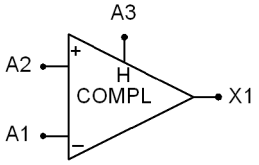
Example — Compare with Hysteresis Middle





Compare with Hysteresis Low

Compare with Hysteresis Low symbol



Programming

Use:Outputs a Boolean value based on a hysteresis comparison between two integer input signals

Function:

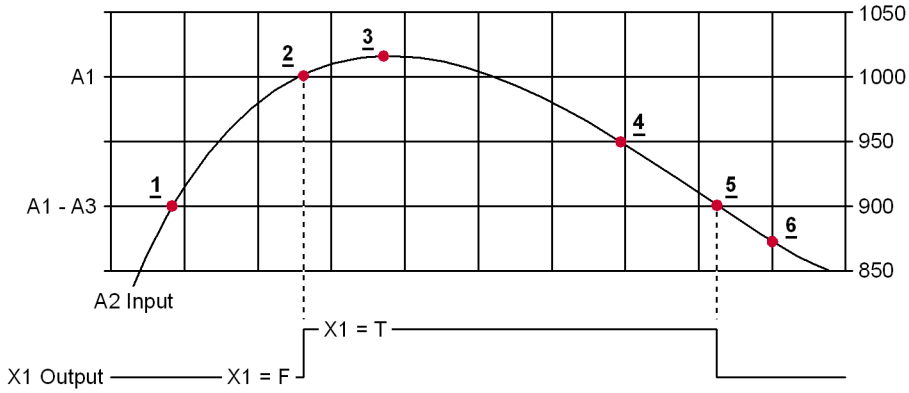
- X1 = True if A2 ≥ A1
- X1 = False if A2 < (A1 – A3)
- X1 = No change if (A1 – A3) ≤ A2 < A1

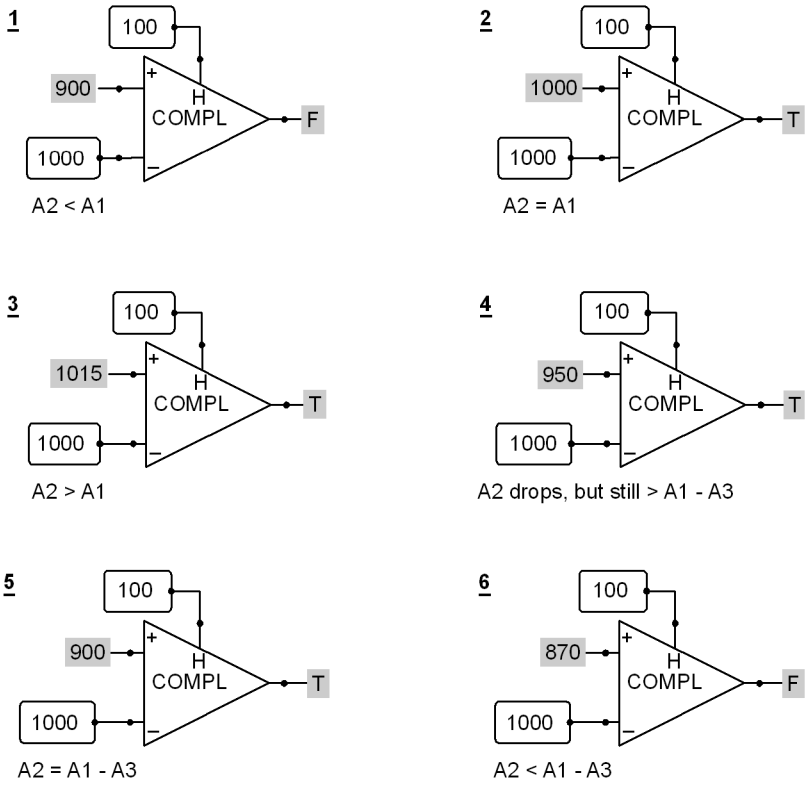
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	BOOL
A2	INT	—	——
A3	UINT	—	——

Programming

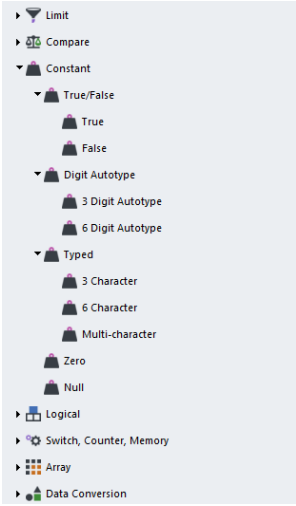
Example — Compare with Hysteresis Low



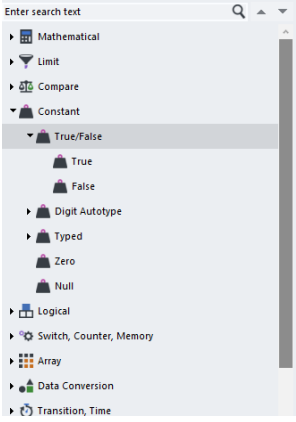


Programming

Constant Menu



True/False Menu



True

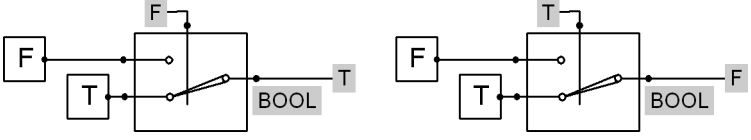
Use: Outputs a Boolean constant True

Function: X1 = True

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	BOOL

Example — True



False

Use: Outputs a Boolean constant False

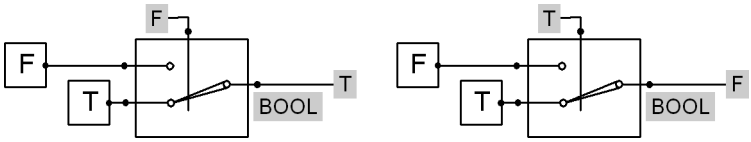
Programming

Function: X1 = False

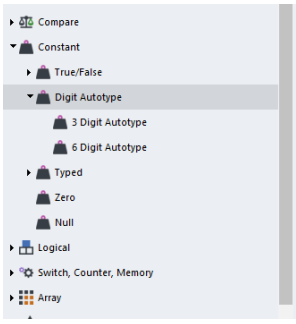
Valid Connections

Pin	Data Type	Pin	Data Type
—	——	X1	BOOL

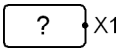
Example — False



Digit Autotype Menu



3 Digit Autotype



Use:

- Outputs a constant value
- Sized for values with up to three digits
- The value of your entry sets the data type (see the table below)
- Only accepts numbers. To output Boolean values, use the **False** and the **True** components

Function: X1 = Auto-typed value of **?**

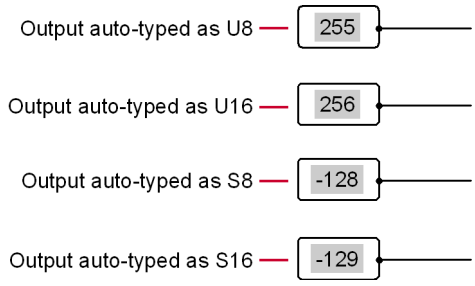
This component accepts values larger than three digits. Larger values that extend beyond this component's boundaries can be hard to read. For larger values, use either the *6 Digit Autotype* component or the *Multi-Character* component.

Data Types

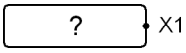
Value Entered	Data Type
0 to +255	U8
–128 to -1	S8
+256 to +65535	U16
–32768 to -129	S16
+65536 to +4294967295	U32
–2147483648 to –32769	S32

Programming

Example — 3 Digit Autotype



6 Digit Autotype



Use:

- Outputs a constant value
- Sized for values with up to six digits
- The value of your entry automatically sets the data type (see the table below)
- Only accepts numbers. To output Boolean values, use the **False** and the **True** components

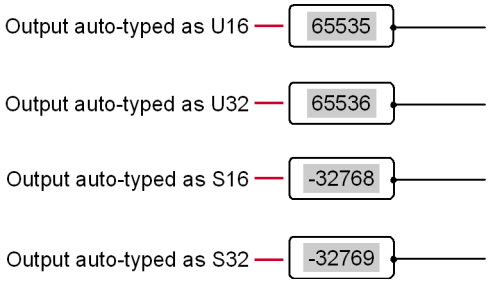
This component accepts values larger or smaller than six digits. Larger values that extend beyond this component's boundaries can be hard to read. For larger values, use the *Multi-Character* component.

Function: X1 = Auto-typed value of **?**

Data Types

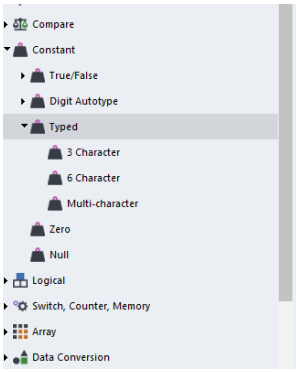
Value Entered	Data Type
0 to +255	U8
–128 to -1	S8
+256 to +65535	U16
–32768 to -129	S16
+65536 to +4294967295	U32
–2147483648 to –32769	S32

Example — 6 Digit Autotype

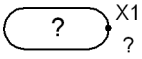


Programming

Typed Menu



3 Character



Use:

- Outputs a constant value
- You must define the data type of the value
- Component is sized for values with up to three digits

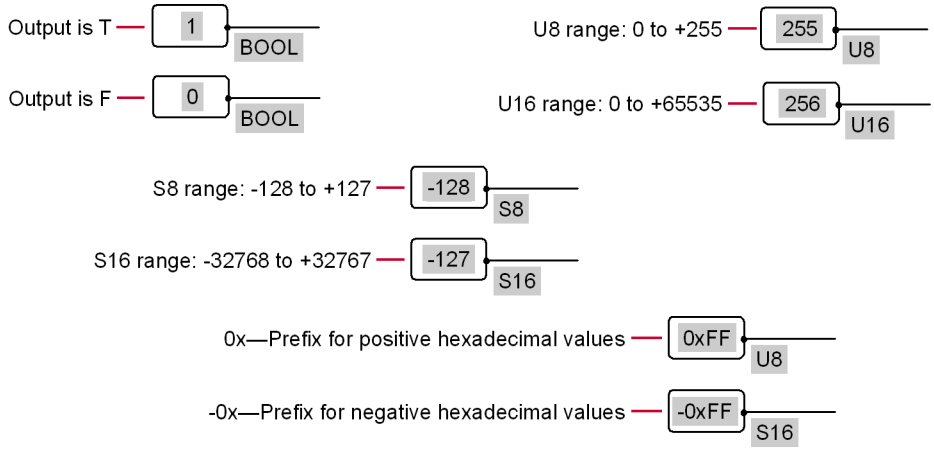
Function: X1 = Value of **?**

This component accepts values larger than three digits. Larger values that extend beyond this component's boundaries can be hard to read. For larger values, use either the *6 Character* component or the *Multi-Character* component.

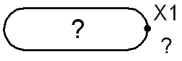
Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	MAIN

Example — 3 Character



6 Character



Use:

Programming

- Outputs a constant value
- You must define the data type of the value
- Component is sized for values with up to six digits

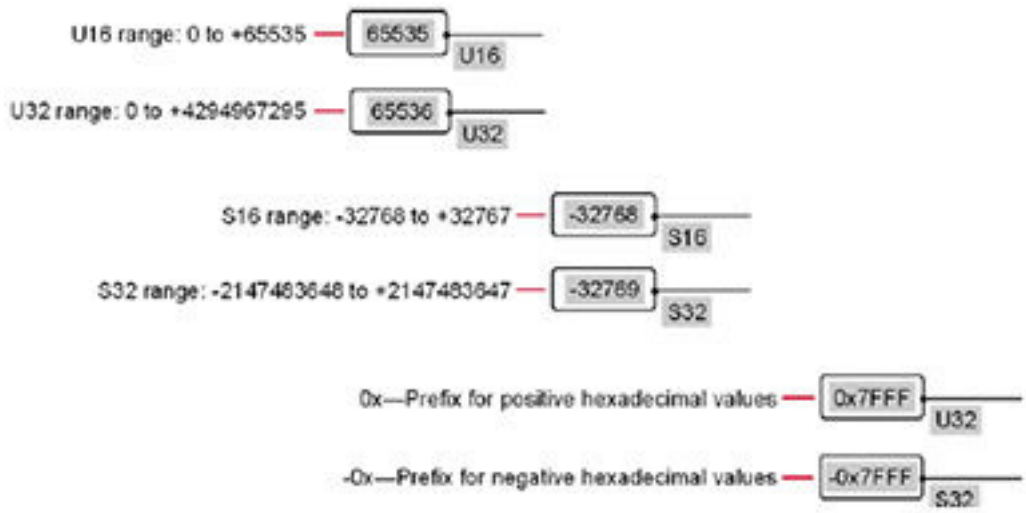
This component accepts values larger than six digits. Larger values that extend beyond this component's boundaries can be hard to read. For larger values, use the *Multi-Character* component.

Function: X1 = Value of **?**

Valid Connections

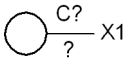
Pin	Data Type	Pin	Data Type
—	——	X1	MAIN

Example — 6 Character



Programming

Multi-Character



Use:

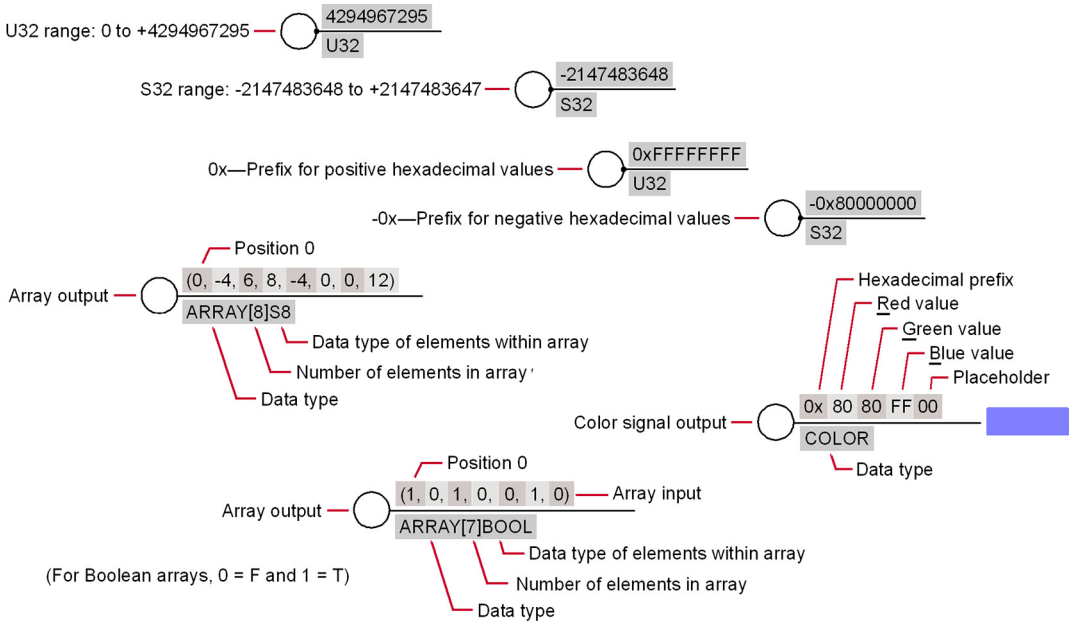
- Outputs a constant value
- You must define the data type of the value
- Use for large, multi-character constants

Function: X1 = C **?**

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	BOOL, U8, U16, U32, S8, S16, S32, COLOR, T, TL, STRING

Example — Multi-character



Zero

Use: Defines a constant value of 0

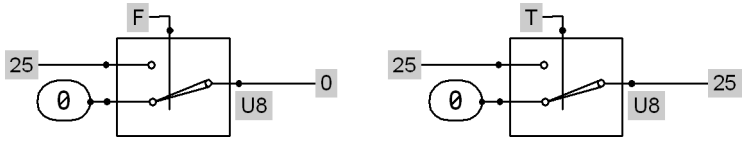
Function: X1 = 0

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	UINT

Programming

Example — Zero



Null

Use:

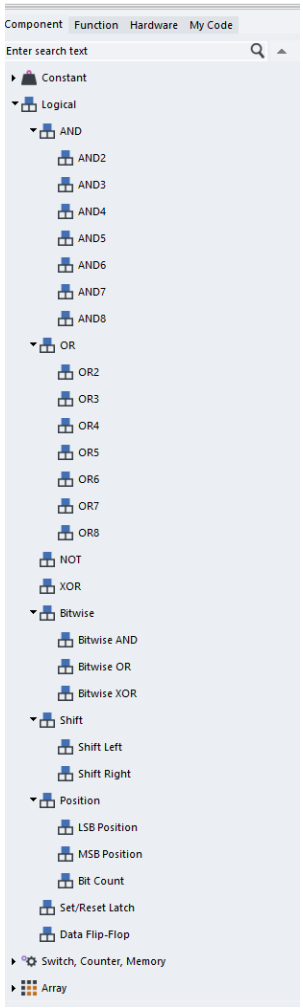
- Indicates that a connection requires no input
- Can provide a NULL value for an input that is not being used

Function: X1 = NULL

Valid connections

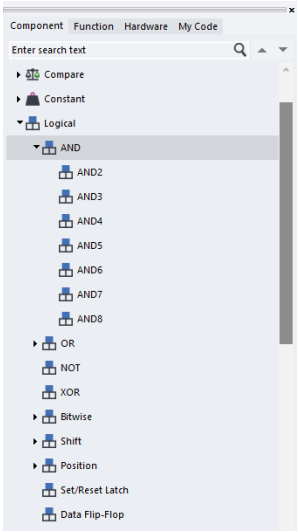
Pin	Data Type	Pin	Data Type
—	——	X1	NULL

Logical Menu



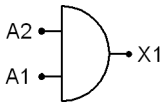
Programming

AND Menu



AND2–AND8

Except for the number of inputs, the **AND2** through **AND8** components function alike. Only the **AND2** component is described here.



Use: AND function for two to eight Boolean signals.

Function: X1 = AND function applied to A1–A8

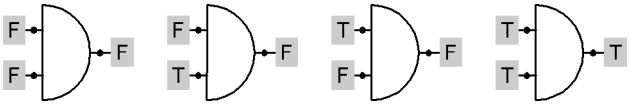
Truth table

A1 Input	A2 Input	X1 Output
False	False	False
True	False	False
False	True	False
True	True	True

Valid connections

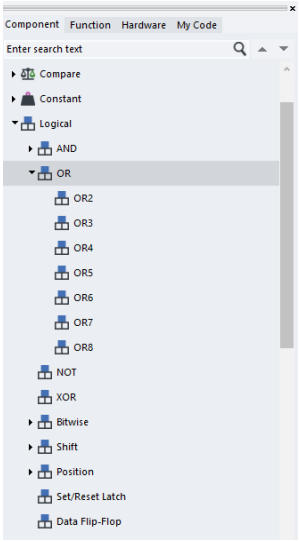
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	BOOL	—	——

Example — AND



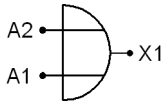
Programming

OR Menu



OR2–OR8

Except for the number of inputs, the **OR2** through **OR8** components function alike. Only the **OR2** component is described here.



Use: OR function for two to eight Boolean signals

Function: X1 = OR function applied to A1–A8

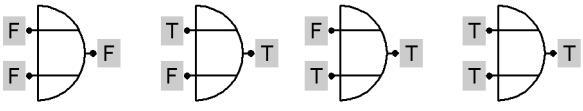
Truth table

A1 Input	A2 Input	X1 Output
False	False	False
True	False	True
False	True	True
True	True	True

Valid connections

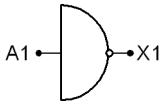
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	BOOL	—	——

Example — OR



Programming

NOT



Use: NOT function for a Boolean signal

Function: X1 = NOT function applied to A1

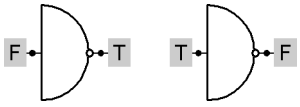
Truth table

A1 Input	X1 Output
False	True
True	False

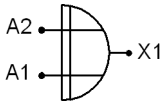
Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL

Example — NOT



XOR



Use: Exclusive OR function for two Boolean signals

Function: XOR function applied to A1 and A2

Truth table

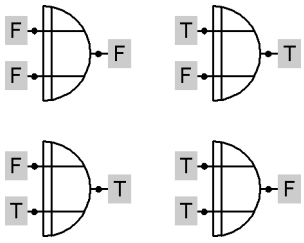
A1 Input	A2 Input	X1 Output
False	False	False
True	False	True
False	True	True
True	True	False

Valid connections

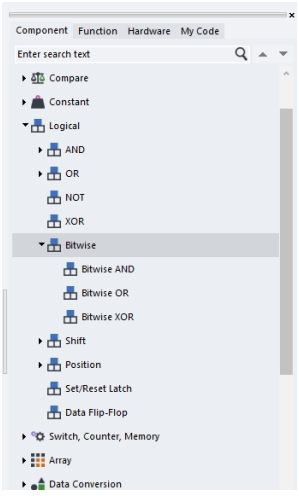
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	BOOL	—	——

Programming

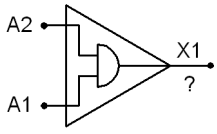
Example — XOR



Bitwise Menu



Bitwise AND



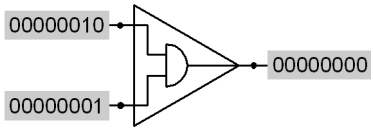
Use: Outputs a mask of the bit states in two integer signals

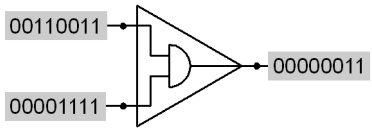
Function: X1 = Bitwise AND output of A1 and A2

Valid connections

Pin	Data Type	Pin	Data Type
A1	UINT	X1	UINT
A2	UINT	—	——

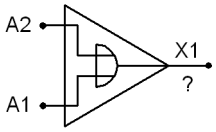
Example — Bitwise AND





Programming

Bitwise OR



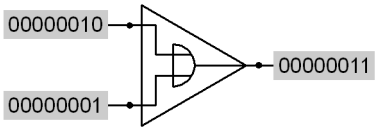
Use: Outputs a merge of the bit states in two integer signals

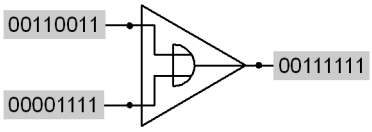
Function: X1 = Bitwise OR output of A1 and A2

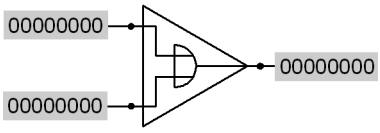
Valid connections

Pin	Data Type	Pin	Data Type
A1	UINT	X1	UINT
A2	UINT	—	——

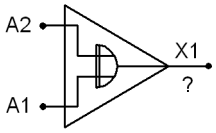
Example — Bitwise OR







Bitwise XOR



Use: Outputs a comparison of the bit states in two integer signals

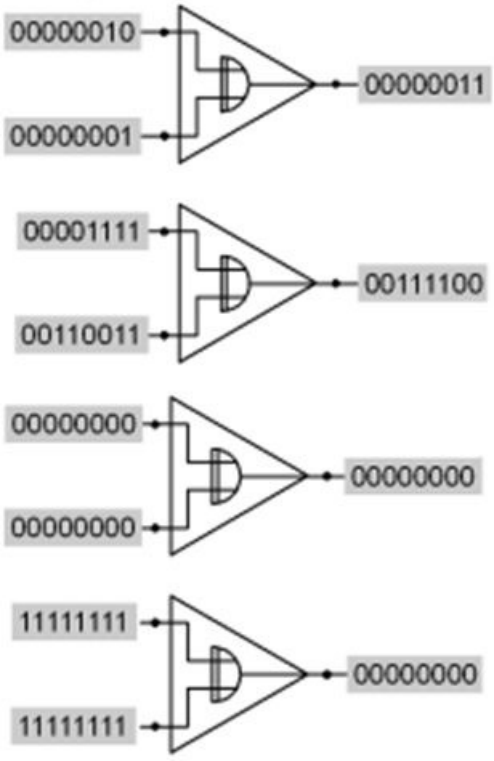
Function: X1 = Bitwise XOR output of A1 and A2

Valid connections

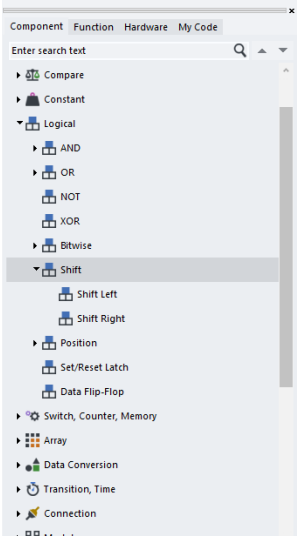
Pin	Data Type	Pin	Data Type
A1	UINT	X1	UINT
A2	UINT	—	——

Programming

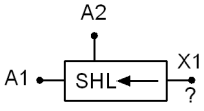
Example — Bitwise XOR



Shift Menu



Shift Left



Use: Shifts bit data in an integer signal

Function: X1 = A1 shifted left by the number of positions indicated by the value of A2

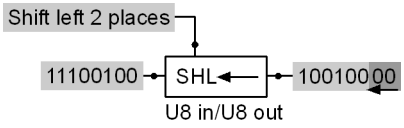
Programming

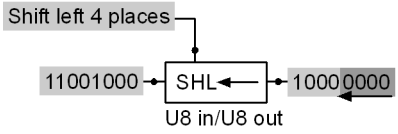
Because the most significant bit in a signed value indicates the value's sign, only use this component with unsigned data types.

Valid connections

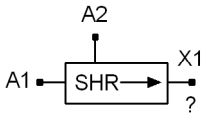
Pin	Data Type	Pin	Data Type
A1	UINT	X1	UINT
A2	UINT	—	——

Example — Shift Left





Shift Right



Use:

- Shifts bit data in an integer signal
- Useful for bit shifting in CAN data

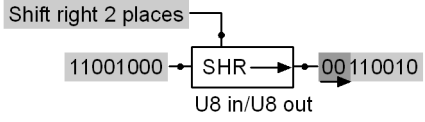
Function: X1 = A1 shifted right by the number of positions indicated by the value of A2

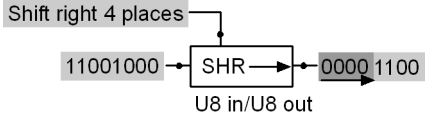
Because the most significant bit in a signed value indicates the value's sign, only use this component with unsigned data types.

Valid connections

Pin	Data Type	Pin	Data Type
A1	UINT	X1	UINT
A2	UINT	—	——

Example — Shift Right





Programming

Bit Count

Use:

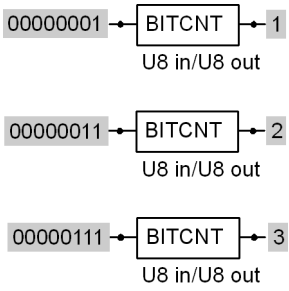
- Outputs the number of bits set to one (1) in an integer signal
- Can count the number of active alarms if alarm states are coded into the byte

Function: X1 = Number of bits with a value of one (1)

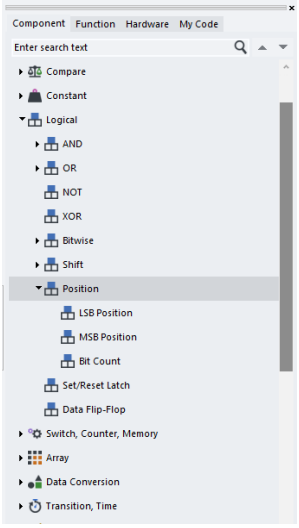
Valid connections

Pin	Data Type	Pin	Data Type
A1	UINT	X1	UINT

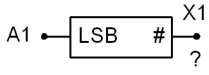
Example — Bit Count



Position Menu



LSB Position



Use:

- Outputs the position of the least significant bit (LSB) set to 1 in an integer signal
- Can indicate the highest priority alarm if alarm states are coded into the byte

Function:

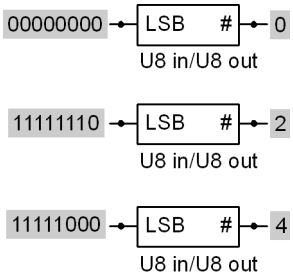
Programming

- X1 = Position (from the right) of the least significant bit with a value of 1
- X1 = 0 if no bits are set to 1

Valid connections

Pin	Data Type	Pin	Data Type
A1	UINT	X1	UINT

Example — LSB Position



MSB Position

Use:

- Outputs the position of the most significant bit (MSB) set to 1 in an integer signal
- Can indicate the highest priority alarm if alarm states are coded into the byte

Function:

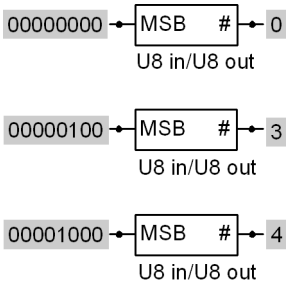
- X1 = Position (from the right) of the most significant bit with a value of 1
- X1 = 0 if no bits are set to 1

- Outputs the position of the most significant bit (MSB) set to 1 in an integer signal
- Can indicate the highest priority alarm if alarm states are coded into the byte

Valid connections

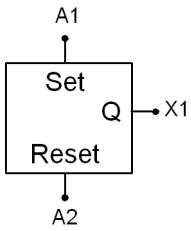
Pin	Data Type	Pin	Data Type
A1	UINT	X1	UINT

Example — MSB Position



Programming

Set/Reset Latch



Use:

- Holds a Boolean signal output
- A True on Set latches Q to True (if Reset is False); Q stays True until Reset becomes True
- Reset has priority over Set

Function:

- X1 = No change if A1 = False and A2 = False
- X1 = Set to True if A1 = True and A2 = False
- X1 = Reset to False if:
  - A1 = False and A2 = True
  - A1 = True and A2 = True
- If X1 = True, it stays True until A2 = True
- If A2 = True, X1 = False

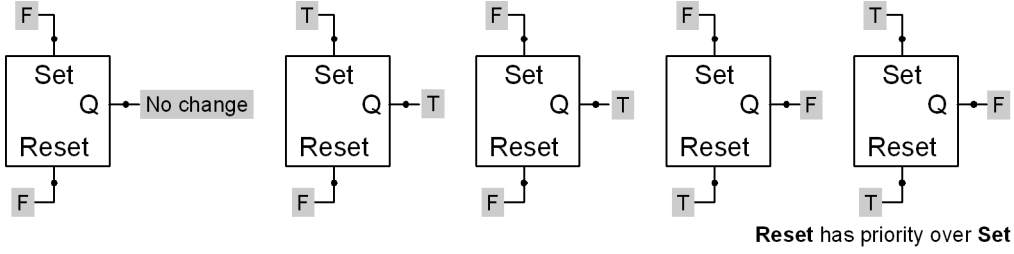
Truth table

A1 Input	A2 Input	X1 Output
False	False	No change
True	False	True
False	True	False
True	True	False

Valid connections

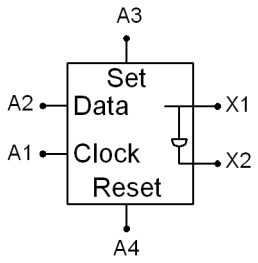
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	BOOL	—	——

Example — Set/Reset Latch



Programming

Data Flip-Flop



Use:

- Holds a Boolean signal output
- Toggles a Boolean signal output on the transition of a Boolean signal input
- Can be used to implement a toggling push button by connecting the X2 output to the A2 input
- Reset input always has priority

Function:

- X1 and X2 = No change if A1, A2, A3, and A4 = False
- X1 = True and X2 = False if A3 = True and A4 = False
- X1 = False and X2 = True if:
  - A3 = False and A4 = True
  - A3 = True and A4 = True
- X1 = False and X2 = True if A1 = Transitions False/True, A2 = False, A3 = False, and A4 = False
- X1 = True and X2 = False if A1 = Transitions False/True, A2 = True, A3 = False, and A4 = False

Truth table

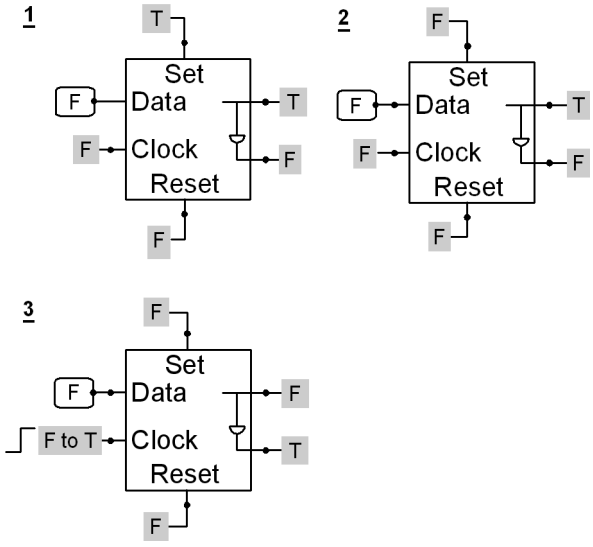
A1 Input	A2 Input	A3 Input	A4 Input	X1 Output	X2 Output
False	False	False	False	No change	No change
——	——	True	False	True	False
——	——	False	True	False	True
——	——	True	True	False	True
False/True transition	False	False	False	False	True
False/True transition	True	False	False	True	False

Valid Connections

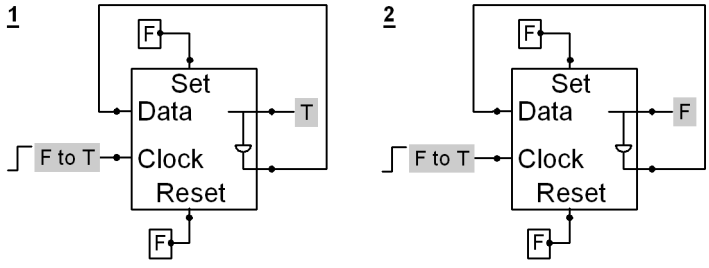
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	BOOL	X2	BOOL
A3	BOOL	—	——
A4	BOOL	—	——

Programming

Example 1—Data Flip-Flop



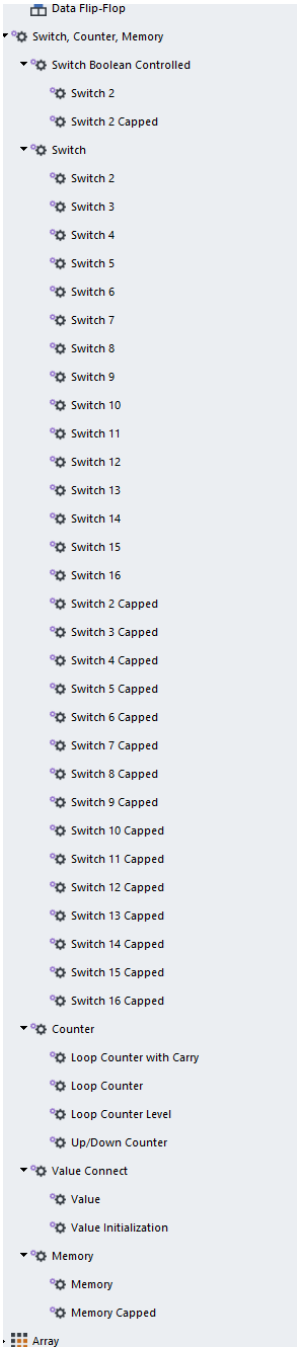
Example 2—Data Flip-Flop



A1—Push-button input. Each button push toggles the X1 state.

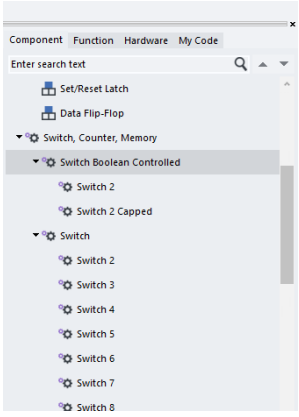
Programming

Switch, Counter, Memory Menu

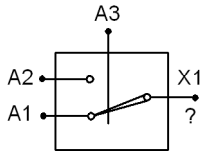


Programming

Switch Boolean Controlled Menu



Switch 2



Use: Switches the output between one of two inputs.

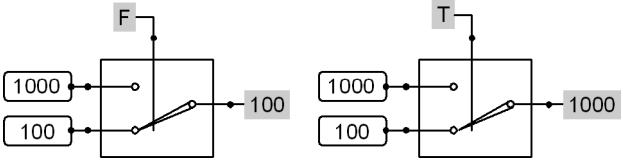
Function:

- X1 = A1 if A3 = False
- X1 = A2 if A3 = True

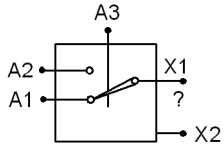
Valid connections

Pin	Data Type	Pin	Data Type
A1, A2	BOOL, INT, TIME, FILE, FONT, PORT, COL, LANG, STRING	X1	BOOL, INT, TIME, FILE, FONT, PORT, COL, LANG, STRING
A3	BOOL	—	——

Example—Switch 2



Switch 2 Capped



Use:

- Switches the output between one of two inputs
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Programming

This component has an unpredictable X2 (True = Overflow) output when used in pages whose **Object** property is **True**. However, its X1 output works correctly on these pages.

Function:

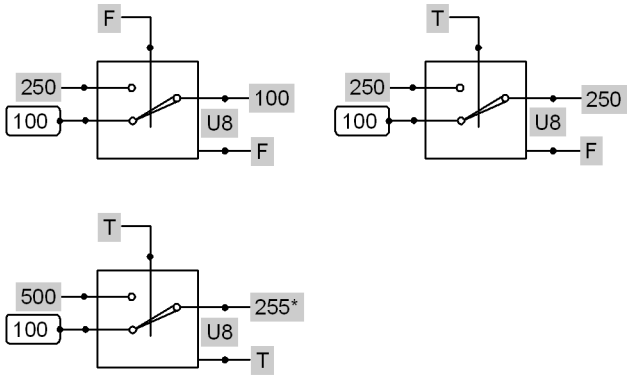
- X1 = A1 if A3 = False
- X1 = A2 if A3 = True
- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

Valid connections

Pin	Data Type	Pin	Data Type
A1, A2	BOOL, INT, TIME, FILE, FONT, PORT, COL, LANG	X1	BOOL, INT, TIME, FILE, FONT, PORT, COL, LANG
A3	BOOL	X2	BOOL

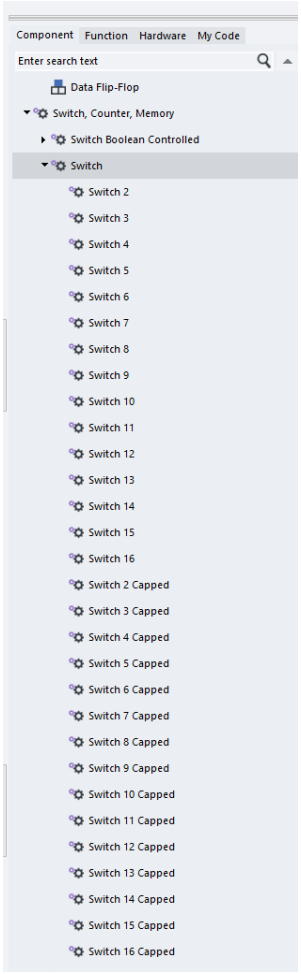
Example—Switch 2 Capped



*255—Max value of the U8 data type

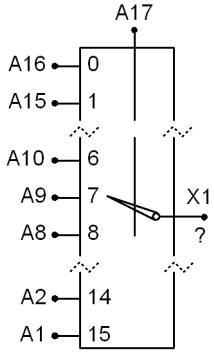
Programming

Switch Menu



Switch 2–16

Except for the number of inputs, the **Switch 2** through **Switch 16** components function alike. Only the Switch 16 component is described here.



Use: Switches the output between 1 of 16 inputs.

Function:

- X1 = A(16 – A17 input value)
- No inputs on this component can float
- All inputs must connect to a value

Programming

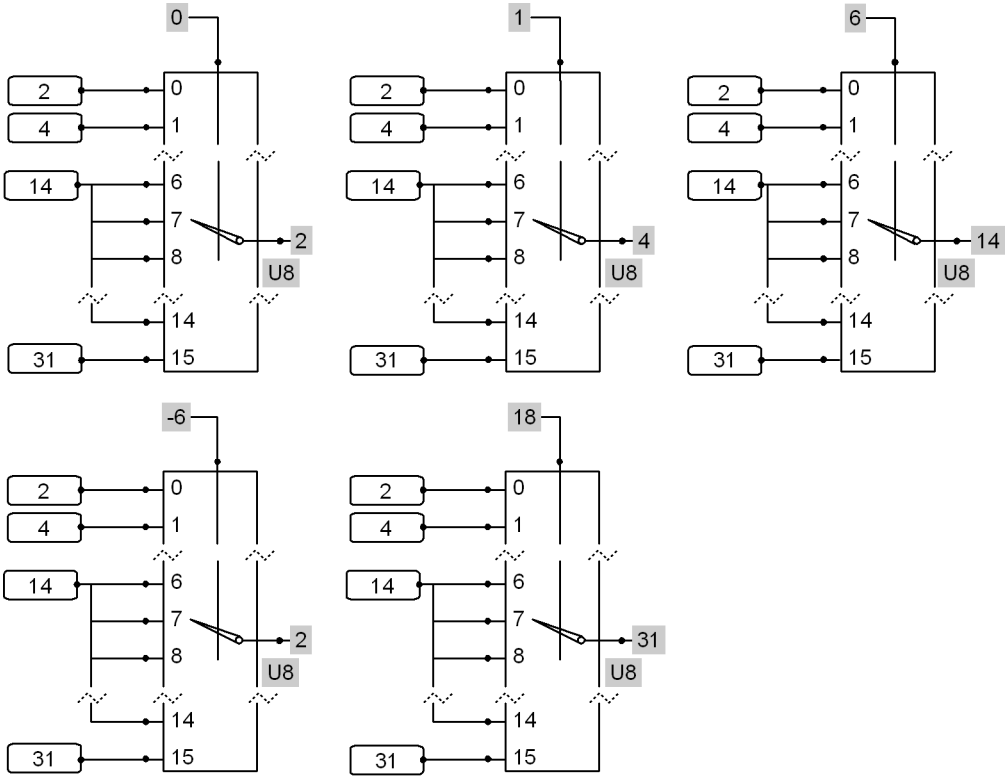
A17 Input—X1 Output

A17 Input	X1 Output	A17 Input	X1 Output	A17 Input	X1 Output
< 0	A16	5	A11	11	A5
0	A16	6	A10	12	A4
1	A15	7	A9	13	A3
2	A14	8	A8	14	A2
3	A13	9	A7	15	A1
4	A12	10	A6	>15	A1

Valid connections

Pin	Data Type	Pin	Data Type
A1–A16	BOOL, INT, TIME, FILE, FONT, PORT, COL, LANG, STRING	X1	BOOL, INT, TIME, FILE, FONT, PORT, COL, LANG, STRING
A17	INT	—	——

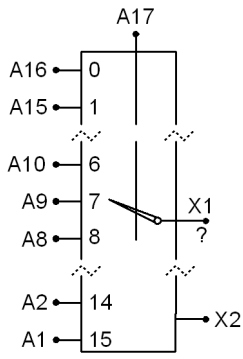
Example—Switch 2–16



Switch 2–16 Capped

Except for the number of inputs, **Switch 2 Capped** through **Switch 16 Capped** components function alike. Only the **Switch 16 Capped** component is described here.

Programming



Use:

- Switches the output between 1 of 16 inputs
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function:

- X1 = A(16 – A17 input value)
- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

- No inputs on this component can float
- All inputs must connect to a value

A17 Input—X1 Output

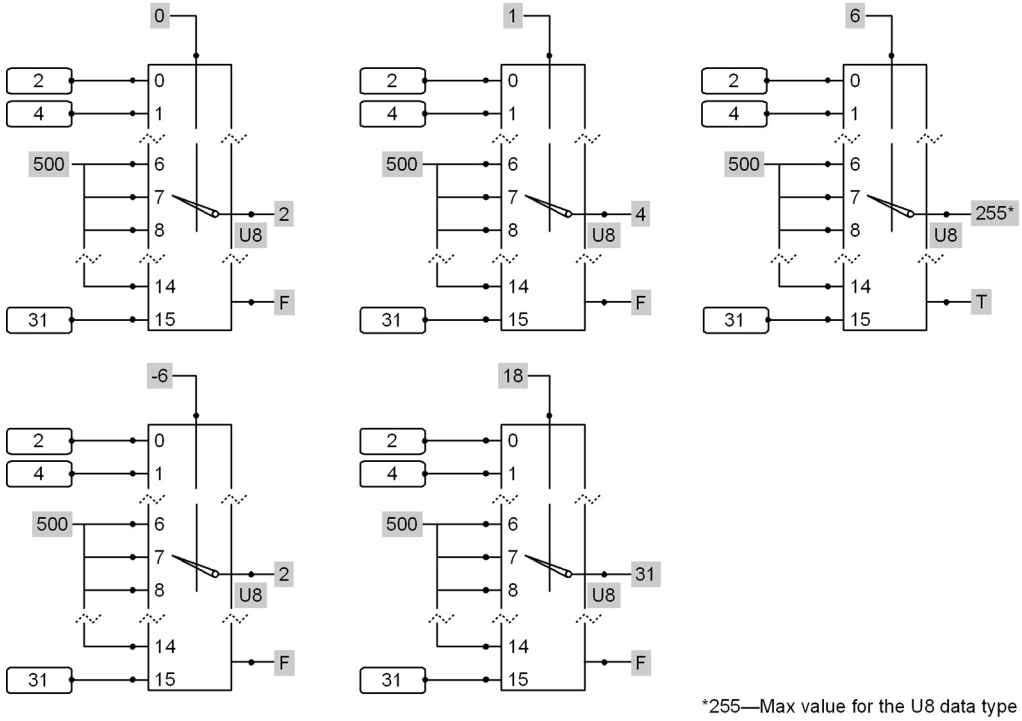
A17 Input	X1 Output	A17 Input	X1 Output	A17 Input	X1 Output
< 0	A16	5	A11	11	A5
0	A16	6	A10	12	A4
1	A15	7	A9	13	A3
2	A14	8	A8	14	A2
3	A13	9	A7	15	A1
4	A12	10	A6	>15	A1

Valid connections

Pin	Data Type	Pin	Data Type
A1–A16	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG	X1	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG
A17	INT	X2	BOOL

Programming

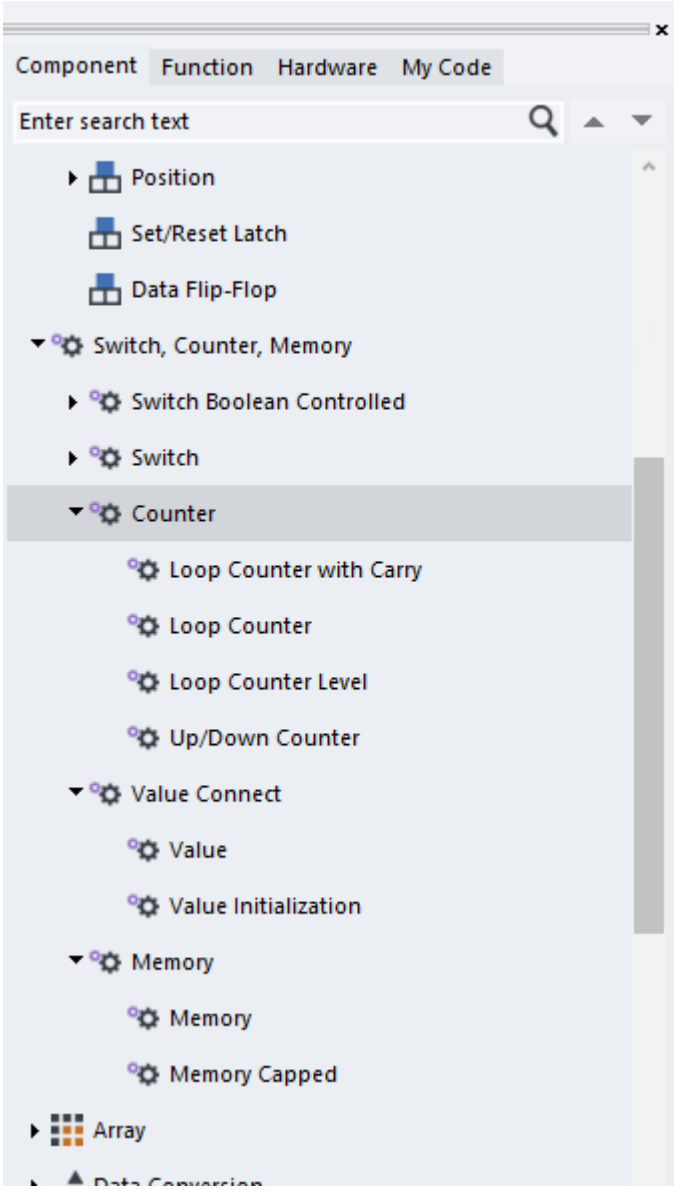
Example—Switch 2–16 Capped



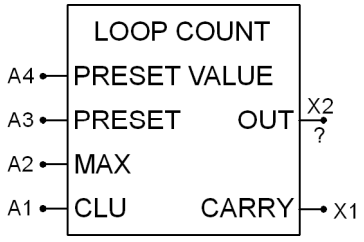
*255—Max value for the U8 data type

Programming

Counter Menu



Loop Counter with Carry



Use:

- Daisy-chain together for large counts
- Use to total seconds, minutes, hours, and days of operating time

Function:

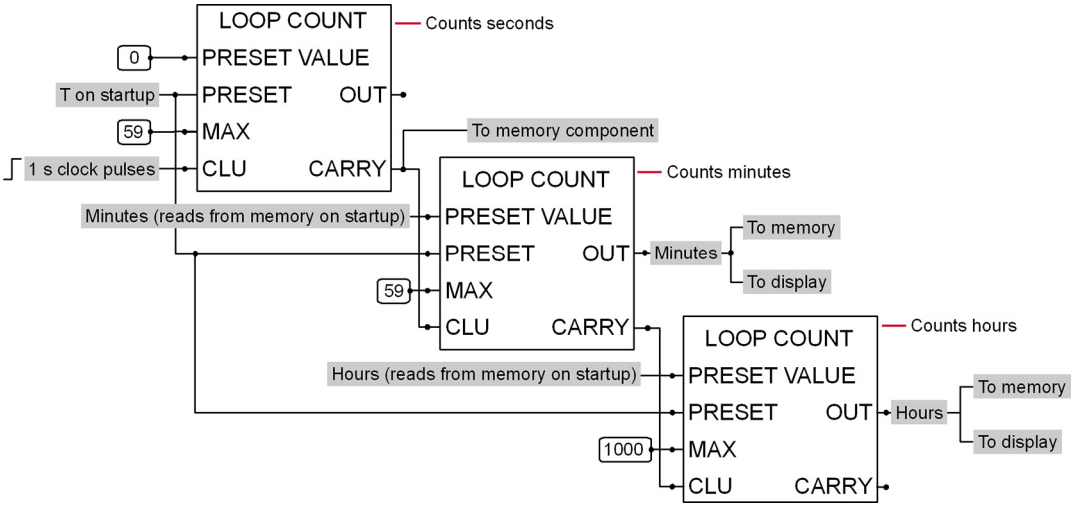
Programming

- A1 = False/True transition increments X2 by 1
- A2 = Maximum loop value
- A3—if:
  - A3 = True, then X2 = A4 (A1 transitions do not increment X2 until A3 becomes False)
  - A3 = False, then A1 transitions increment X2
- A4 = Preset value
- X1 = True when X2 = A2 and A1 transitions False/True; stays True for one clock cycle until A1 again transitions False/True
- X2:
  - Count out
  - Resets to 0 when X1 = A2 and A1 transitions False/True

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	UINT	X2	UINT
A3	BOOL	—	——
A4	UINT	—	——

Example—Loop Counter with Carry

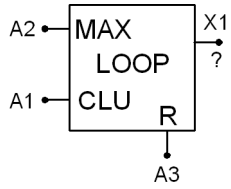


This example shows a timer that totals seconds, minutes, and hours.

- Every 60 seconds, the **LOOP COUNT** seconds component outputs a True **CARRY** that:
  - Increments the **LOOP COUNT** minutes component by one.
  - Writes **OUT** values from the **LOOP COUNT** minute and hour components to memory.
- Every 60 minutes, the **LOOP COUNT** minutes component outputs a True **CARRY**. This output increments the **LOOP COUNT** hours component by one.
- On start up, a True **PRESET**:
  - Applies a **PRESET VALUE** of **0** to the **LOOP COUNT** seconds component.
  - Applies a **PRESET VALUE** read from memory to the **LOOP COUNT** minutes and hours components. (When this example is first downloaded, all **PRESET** inputs need to be manually initialized to **False**. This example does not show this logic.)

Programming

Loop Counter



Use:

- Outputs a count based on False to True transitions

Function:

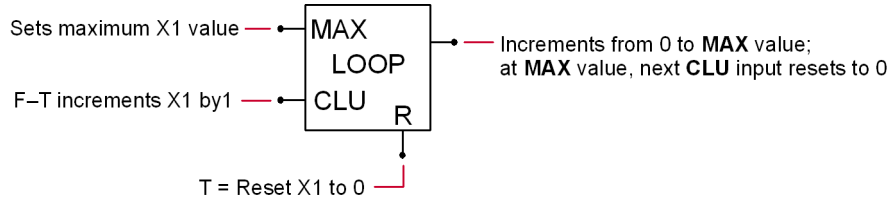
- A1: Each False to True transition increments X1 by 1
- A2: Maximum value of X1

  If A2 is larger than the type maximum of X1, then the type maximum of X1 is used instead of A2.
- A3: If True, then X1 is reset to 0
- X1: Value between 0 and the maximum value
  - Resets to 0 if A3 is True
  - Otherwise, set to A2 if A2 is less than X1
  - Otherwise, if the increment condition is in effect (See A1), then increments by 1 if X1 is less than maximum value or resets to 0
  - Otherwise, no change

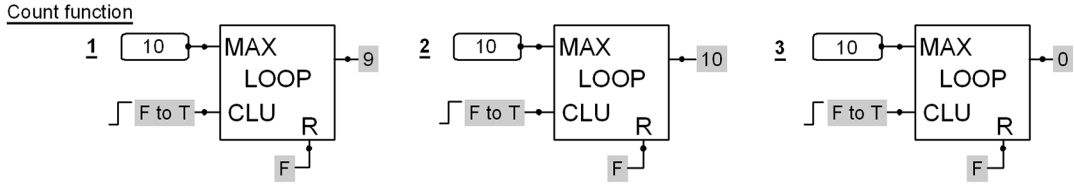
Valid connections

Pin	Data type	Pin	Data type
A1	BOOL	X1	INT
A2	UINT	—	——
A3	BOOL	—	——

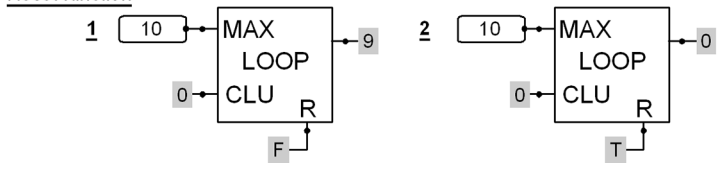
Example—Loop Counter



Count function

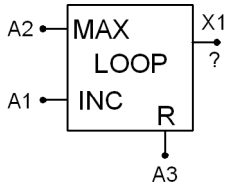


Reset function



Programming

Loop Counter Level



Use:

- Outputs a count based on number of executions of the component

Function:

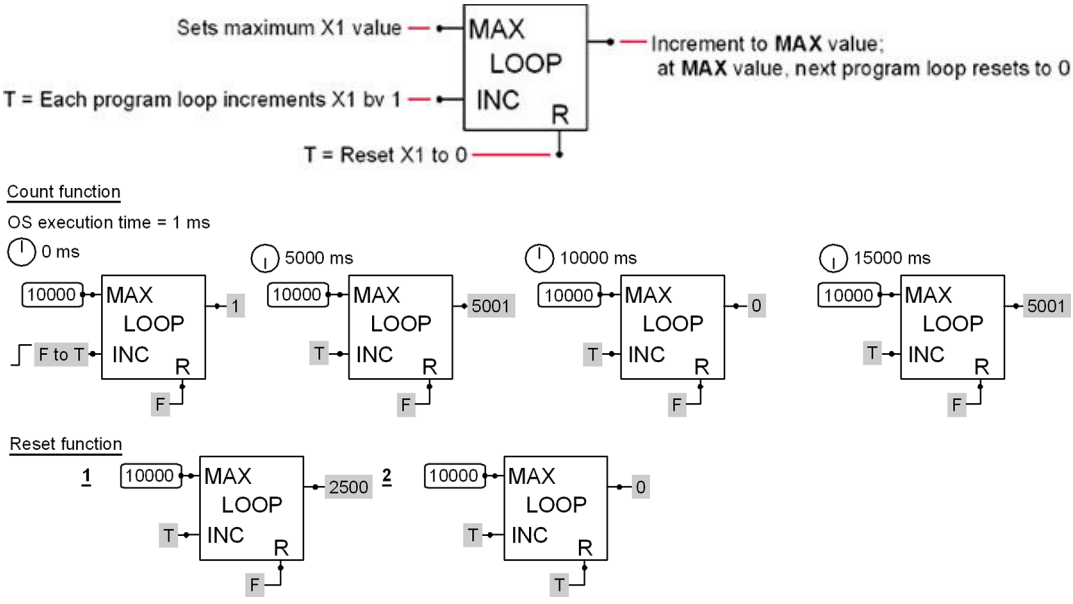
- A1: If True, then increment X1 by 1
- A2: Maximum value of X1

  If A2 is larger than the type maximum of X1, then the type maximum of X1 is used instead of A2.
- A3: If True, then X1 is reset to 0
- X1: Value between 0 and the maximum value
  - Resets to 0 if A3 is True
  - Otherwise, set to A2 if A2 is less than X1
  - Otherwise, if the increment condition is in effect (See A1), then increments by 1 if X1 is less than maximum value or resets to 0
  - Otherwise, no change

Valid connections

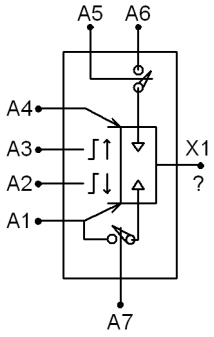
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	INT
A2	UINT	—	——
A3	BOOL	—	——

Example—Loop Counter Level



Programming

Up/Down Counter



Use: Counts the transitions of a Boolean signal

Function:

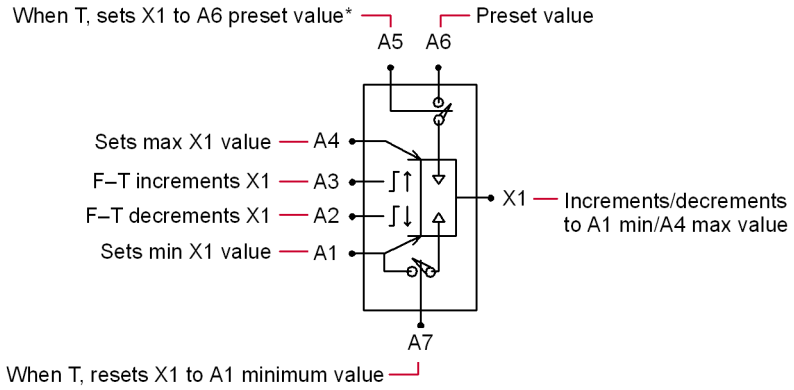
- X1 is a value that goes from the A1 minimum value to the A4 maximum value
- X1 = A1 if A7 = True
- X1 = A6 if A5 = True and A7 = False
- X1 increments each time A3 transitions from False/True until X1 = A4 maximum value
- X1 decrements each time A2 transitions from False/True until X1 = A1 minimum value
- If the A6 preset enable value is greater than the A4 maximum value, then X1 = A4
- If the A6 preset enable value is less than the A1 minimum value, then X1 = A1
- A7 (Set minimum) = True has priority over A5 (Preset enable) = True

Valid connections, Pin functions

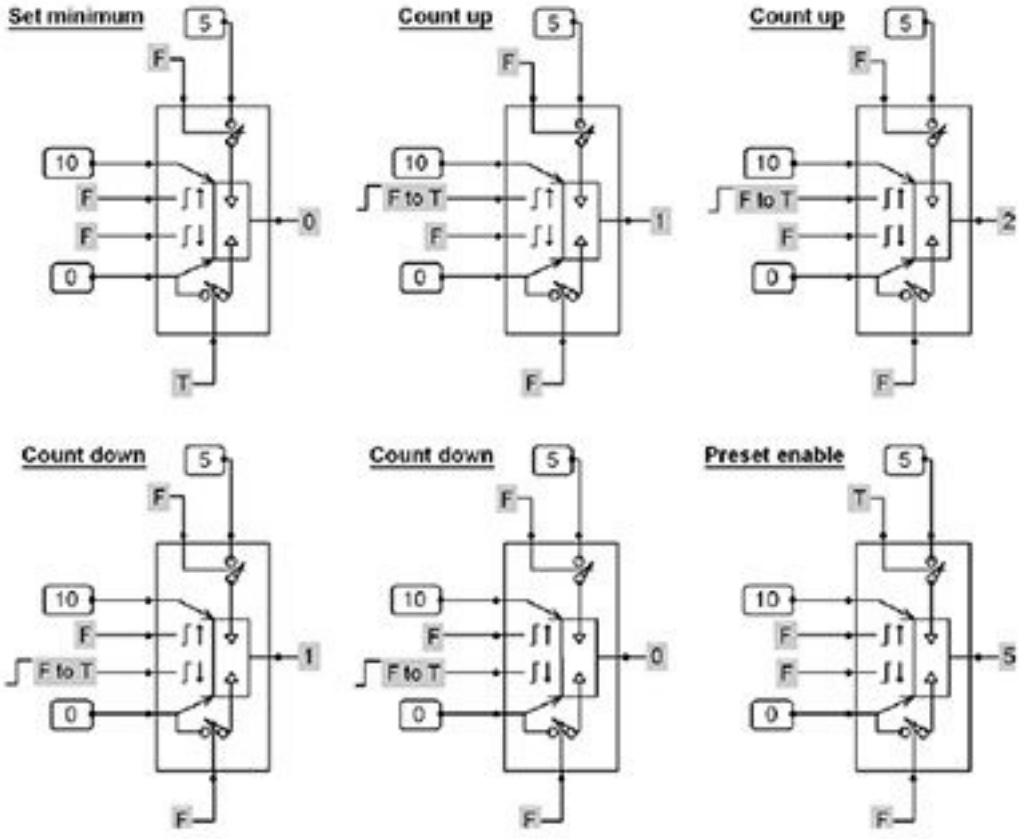
Pin	Data Type	Function
A1	INT	Minimum count value
A2	BOOL	Decrement count
A3	BOOL	Increment count
A4	INT	Maximum count value
A5	BOOL	Preset count enable
A6	INT	Preset count value
A7	BOOL	Set minimum count value
X1	INT	Count output

Programming

Example—Up/Down Counter

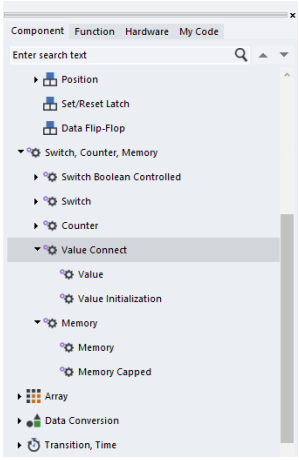


*If A5 is T:
· Sets X1 to A1 min value when A6 value is less than A1 value.
· Sets X1 to A4 max value when A6 value is greater than A4 value.

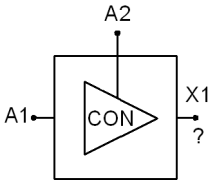


Programming

Value Connect Menu



Value



Use:

- If A2 = True, then A1 = X1
- If A2 = False, then X1 value does not change
- Switches between signals when used in a group (see the following example)
- Samples and holds a value
- Wire the X1 outputs of these components together to create an OR function

Function:

- When using multiple components, whichever one has its A2 input active (A2 = True) will have its input connected to the output
- If more than one A2 input is active (A2 = True), the value of the last one in the execution order is valid, see Example 3—Value below.

Truth table

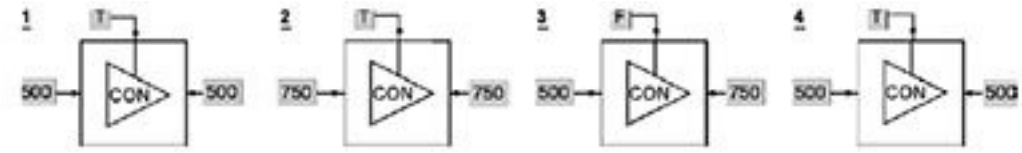
A2 Input	X1 Output
False	No change
True	A1

Valid connections

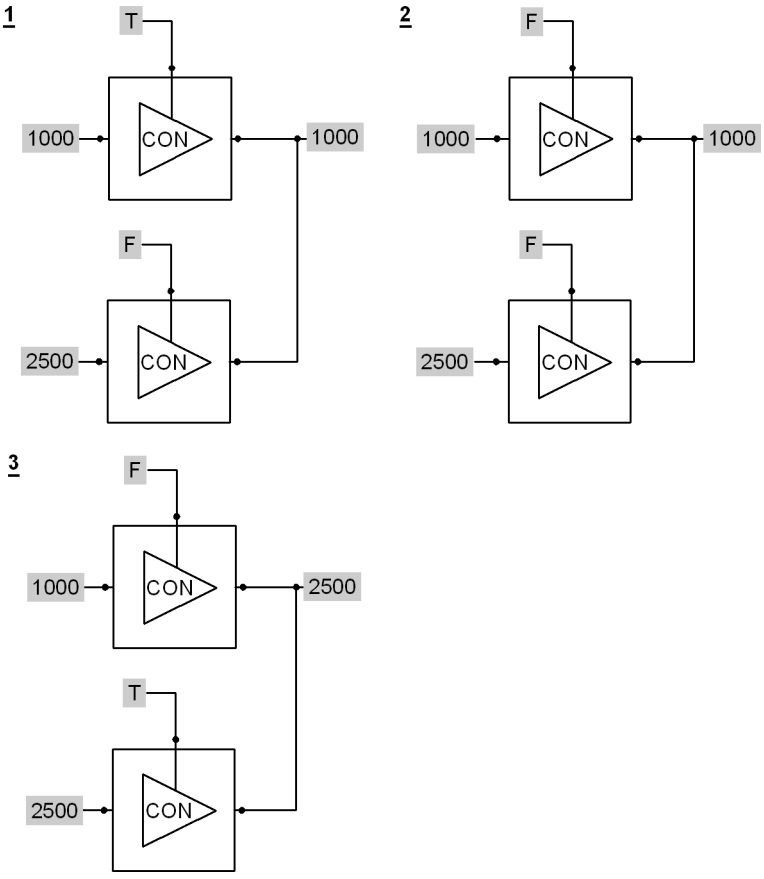
Pin	Data Type	Pin	Data Type
A1	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG	X1	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG
A2	BOOL	—	——

Programming

Example 1—Value

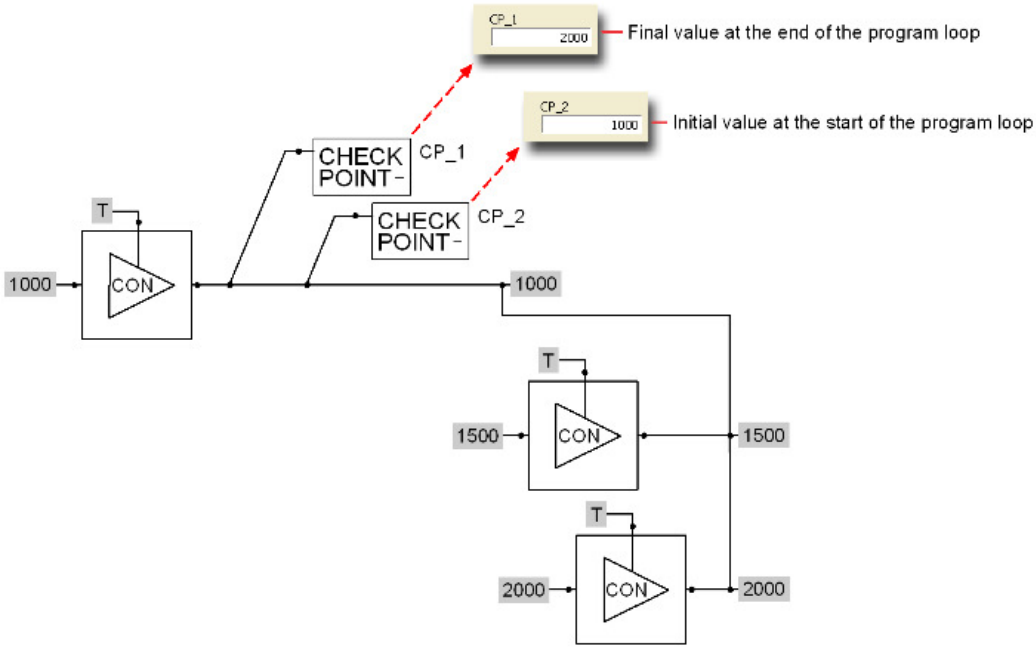


Example 2—Value



Programming

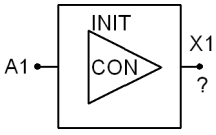
Example 3—Value



The first checkpoint that you connect to a connect net gets "ownership" of that net.

- Any checkpoints that follow the first checkpoint only get copies of the net.
- Net copies are sampled in the places where you place the following checkpoints.
- The Service Tool program receives values from the ECU between program loops.
- The checkpoint that owns the net receives the net's final value at the end of each program loop.
- If you want the initial value of a net, then you must manually create a buffer by connecting two checkpoints after the first connect net output. The first check point will have the final value of the net. The second checkpoint will have the initial value of the net.

Value Initialization



Use:

- Initialize values at the start of each program loop

Function:

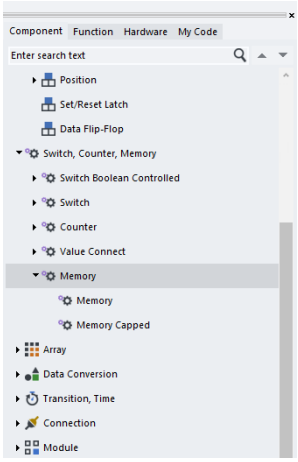
- Always executes first, regardless of its position in the application
- A1 = X1 at the start of each program loop
- Input to A1 must be from a constant component
- Connect the output of this component to the outputs of other **Value Connect** components
- When **Value Initialization** components are used in a group, their input values must be identical

Valid connections

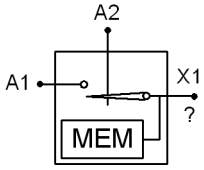
Pin	Data Type	Pin	Data Type
A1	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG	X1	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG

Programming

Memory Menu



Memory



Use:

- Samples a signal value and keeps this value after the signal changes

Function:

- X1 = A1 when A2 = True
- X1 = No change if A2 = False
- You cannot wire together multiple outputs from this component
- Use the **Value** component (see the *Value* component) when you need to wire multiple outputs together

Truth table

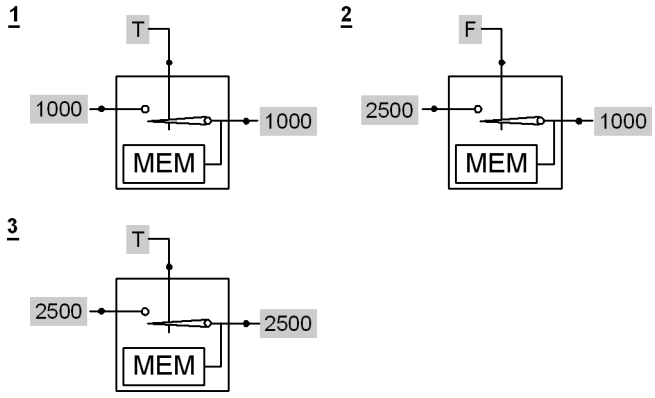
A2 Input	X1 Output
False	No change
True	A1

Valid connections

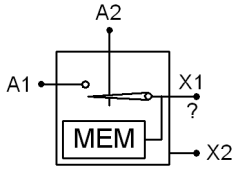
Pin	Data Type	Pin	Data Type
A1	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG	X1	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG
A2	BOOL	—	——

Programming

Example—Memory



Memory Capped



Use:

- Samples a signal value and keeps this value after the signal changes
- Clamps its X1 output if it overflows and sets its X2 output to True to indicate an overflow condition
- Boolean outputs on capped components can be wired together; an overflow condition outputs a True on the net

Function:

- X1 = A1 when A2 = True
- X1 = No change if A2 = False
- If X1 does not overflow, then X2 = False
- If X1 overflows, then:
  - X2 = True
  - X1 = Clamps at the minimum or maximum value of its data type
- X2 resets to False at the start of each program loop

An overflow condition clamps X1 at either its minimum or maximum data type value. Input values determine whether X1 clamps at its minimum or maximum data type value.

- You cannot wire together multiple X1 outputs from this component
- Use the **Value** component (see the *Value* component) when you need to wire multiple X1 outputs together

Truth table

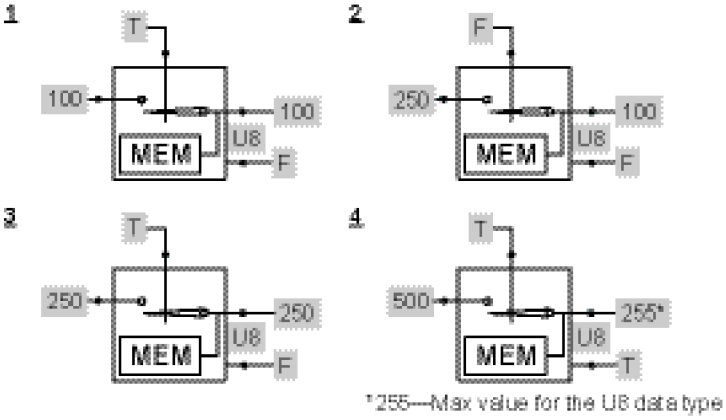
A2 Input	X1 Output
False	No change
True	A1

Programming

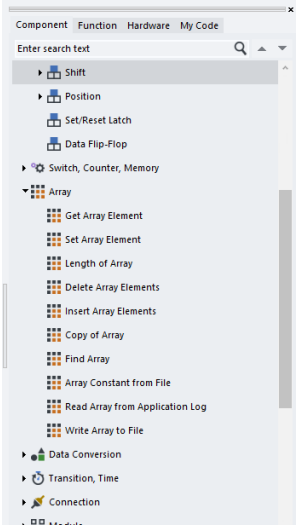
Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG	X1	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG
A2	BOOL	X2	BOOL

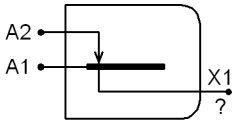
Example—Memory Capped



Array Menu



Get Array Element



Use: Output the value of an element in an array.

Function:

- A1 = Array input
- A2 = Position of element to be output

  The first element in an array is in position 0.
- X1 = Copied element

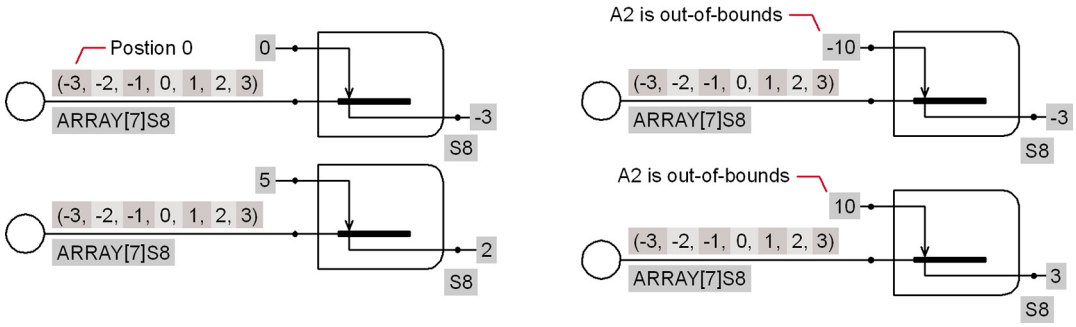
Programming

- X1 = Position 0 array if A2 is negatively out-of-bound
- X1 = Last array if A2 is positively out-of-bounds
- An array can have a maximum of 32,767 elements.
- Different array data types on input and output pins can produce overflow on the X1 output.

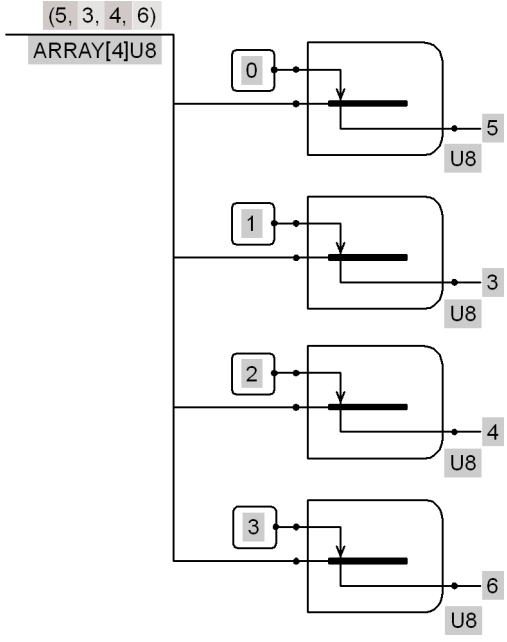
Valid connections

Pin	Data Type	Pin	Data Type
A1	ARRAY	X1	BOOL, INT
A2	U8, S8, S16	—	——

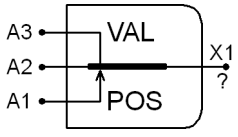
Example 1—Get Array Element



Example 2—How to extract individual elements from an array.



Set Array Element



Use:

Programming

- Set an element in an array to a defined value

Function:

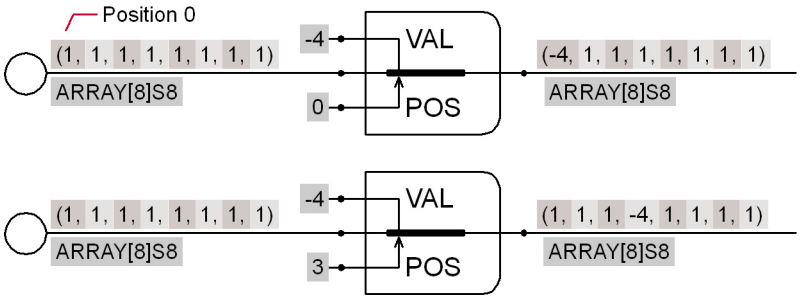
- A1 = Position of the element to be defined

  The first element in an array is in position 0
- A2 = Array with element to be set
- A3 = Value of element to be set
- X1 = Array with changed element
- X1 = A2 if A1 (the set position) is positively or negatively out-of-bounds
- An array can have a maximum of 32,767 elements
- Different arraydata types on input and output pins can produce overflow on the X1 output

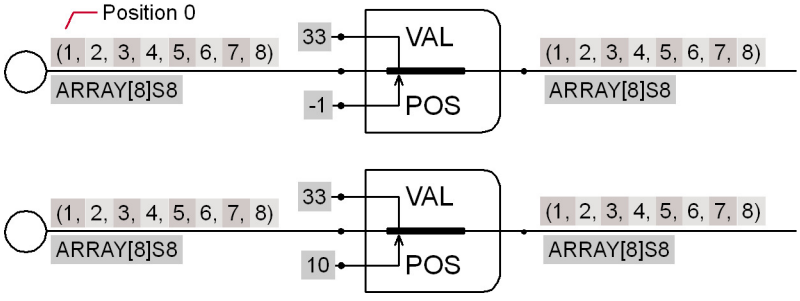
Valid connections

Pin	Data Type	Pin	Data Type
A1	U8, S8, S16	X1	ARRAY
A2	ARRAY	—	——
A3	INT, BOOL	—	——

Example 1—Set Array Element

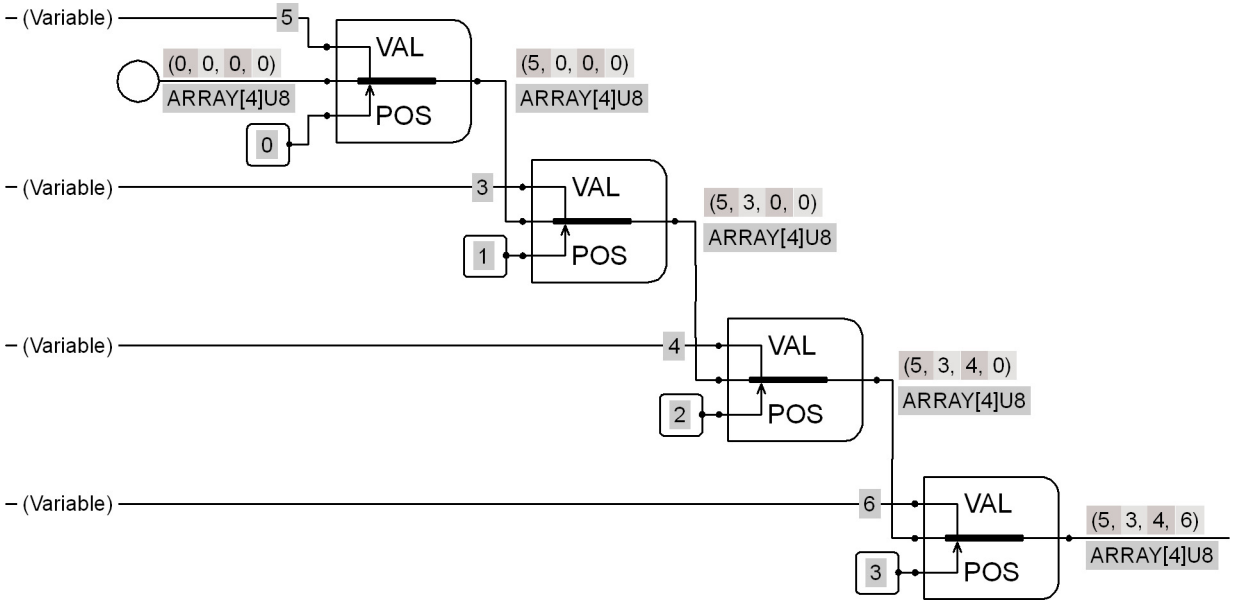


*Example 2—shows what happens when the **A1 Pos** input is out-of-bounds.*

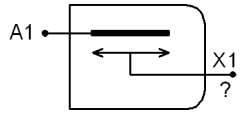


Programming

Example 3—shows how to construct an array that contains variables.



Length of Array



Use:

- Outputs the number of elements in an array

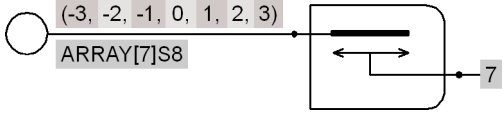
Function:

- A1 = Array
- X1 = Number of elements in array
- An array can have a maximum of 32,767 elements
- Different array data types on input and output pins can produce can produce overflow on the X1 output

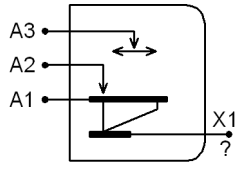
Valid connections

Pin	Data Type	Pin	Data Type
A1	ARRAY	X1	UINT

Example—Length of Array



Delete Array Elements



Programming

Use: Delete elements from an array

Function:

- A1 = Array from which elements are deleted
- A2 = Starting position of first element deleted

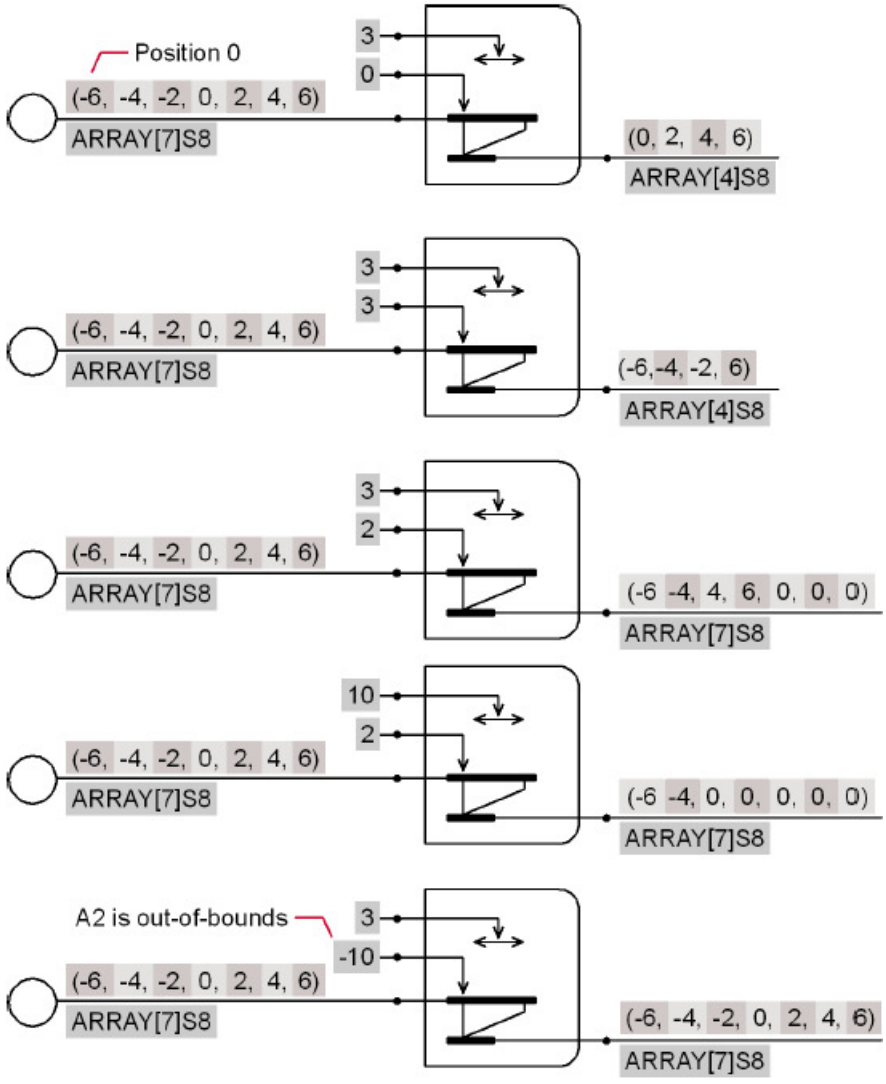
  The first element in an array is in position 0
- A3 = Number of elements deleted from array
- X1 = Array after elements are deleted
- X1 = A1 if A2 (the start delete position) is positively or negatively out-of-bounds
- An array can have a maximum of 32,767 elements
- Different array data types on input and output pins can produce overflow on the X1 output
- Input and output arrays must be the same data type

Valid connections

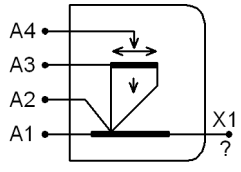
Pin	Data Type	Pin	Data Type
A1	ARRAY	X1	ARRAY
A2	U8, S8, S16	—	——
A3	INT	—	——

Programming

Example—Delete Array Elements



Insert Array Elements



Use:

- Insert additional elements into an array

Function:

- A1 = Array into which new elements are to be inserted
- A2 = Starting position from which new elements are inserted

  The first element in an array is in position 0
- A3 = Array with elements to be inserted
- A4 = Number of elements to be inserted

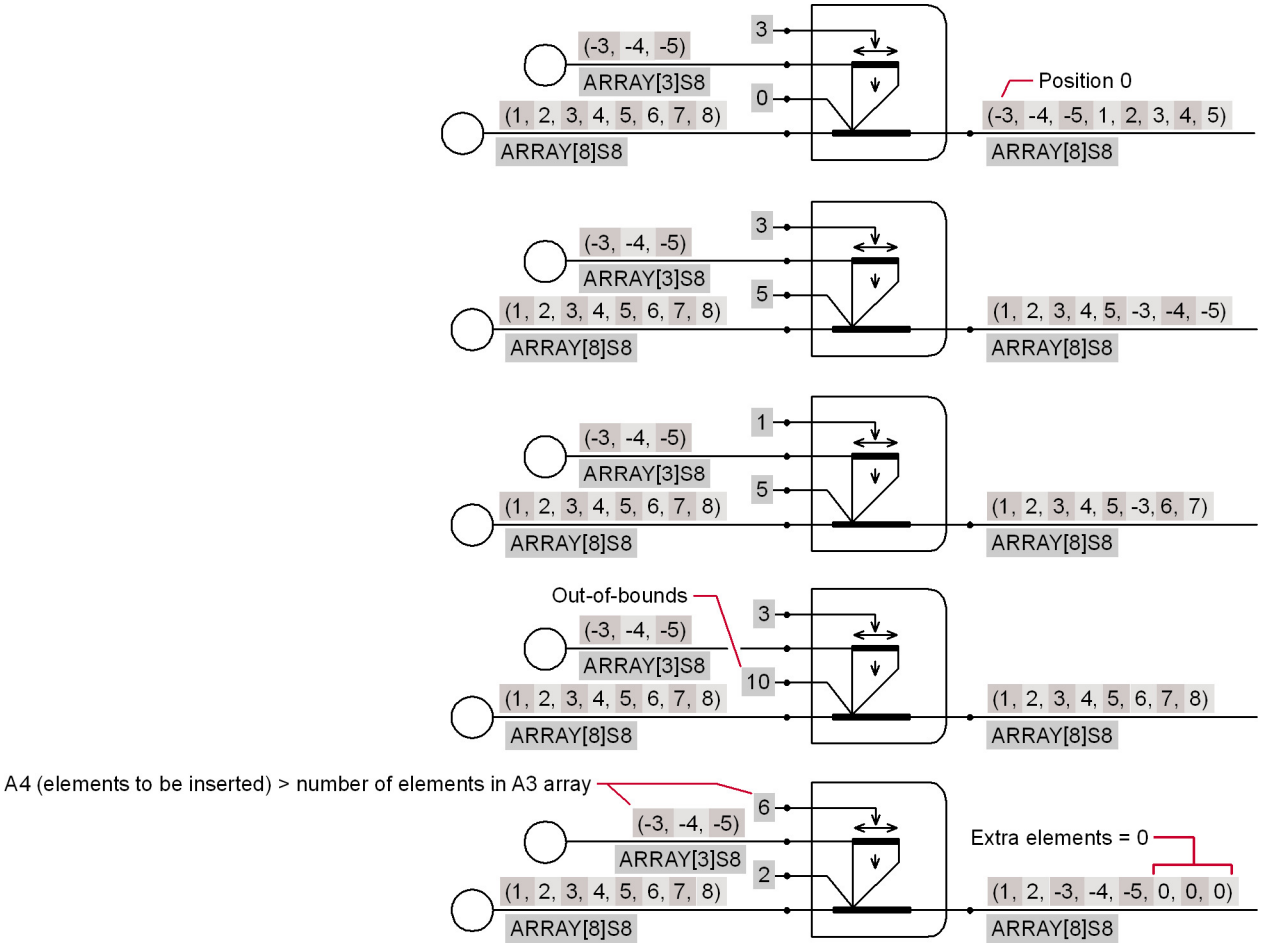
Programming

- X1 = Array after elements are inserted
- X1 = A1 if A2 (the insert position) is out-of-bounds
- If A4 (the number of elements inserted) > number of elements in the A3 array, then the component outputs the extra elements from A4 as zeros
- An array can have a maximum of 32,767 elements
- Different array data types on input and output pins can produce can produce overflow on the X1 output
- Input and output arrays must be the same data type

Valid connections

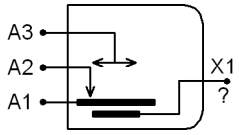
Pin	Data Type	Pin	Data Type
A1	ARRAY	X1	ARRAY
A2	U8, S8 S16	—	——
A3	ARRAY	—	——
A4	INT	—	——

Example—Insert Array Elements



Programming

Copy of Array



Use: Copy and output elements from an array

Function:

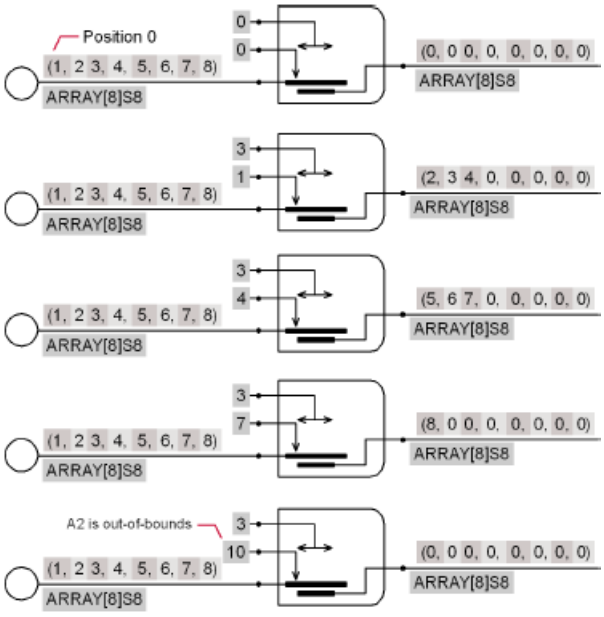
- A1 = Array with elements to be copied
- A2 = Position of first element to be copied

  The first element in an array is in position 0
- A3 = Number of elements to be copied
- X1 = Copied elements
- An array can have a maximum of 32,767 elements
- Different array data types on input and output can produce overflow on the X1 output

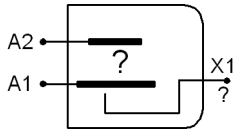
Valid connections

Pin	Data Type	Pin	Data Type
A1	ARRAY	X1	ARRAY
A2	U8, S8, S16	—	——
A3	UINT	—	——

Example—Copy of Array



Find Array



Use:

Programming

- Finds the position of a matching element or the starting position of elements that match

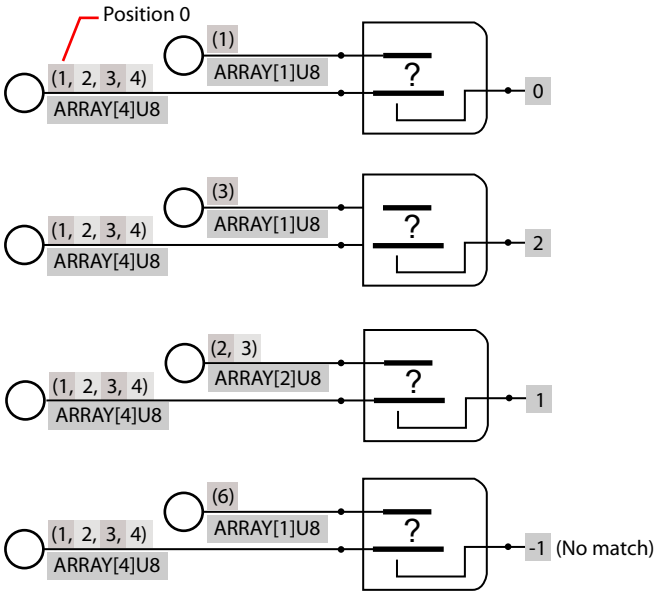
Function:

- A1 = Array with elements to be matched
- A2 = Array elements to be matched
- X1 = Position of matching element or the starting position of elements that match
- If no match, then X1 = -1

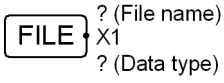
Valid connections

Pin	Data Type	Pin	Data Type
A1	ARRAY	X1	S16
A2	ARRAY	—	——

Example—Find Array



Array Constant from File



Use:

- Input data from text (***.txt) file
- Data must be comma delimited; only one array per text file

Function:

- X1 = Array with elements set to comma delimited values from file:
  - Values with leading zero will be interpreted as octal values
  - Values prefixed by 0x will be interpreted as hexadecimal values
  - All other values will be interpreted as decimal values
- **?** = Name of array

Programming

- The array name must start with a capital letter
- The name of the array must not contain a period (.)

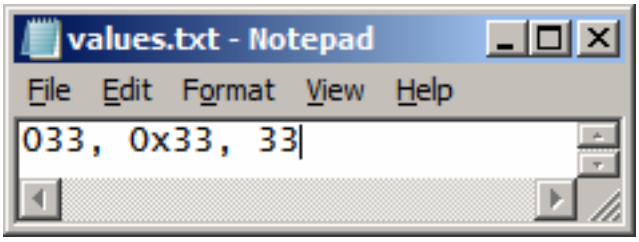
Examples

Correct an array name	Not correct an array name
Values	values
Values1	Values.1

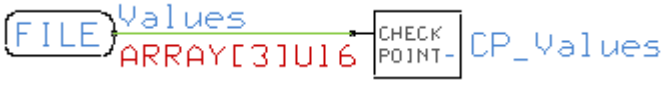
Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	ARRAY

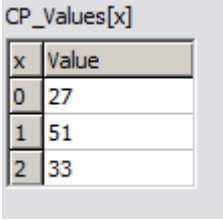
Example file containing value 33 written in octal, hexadecimal and decimal form



Simple GUIDE drawing example using the array

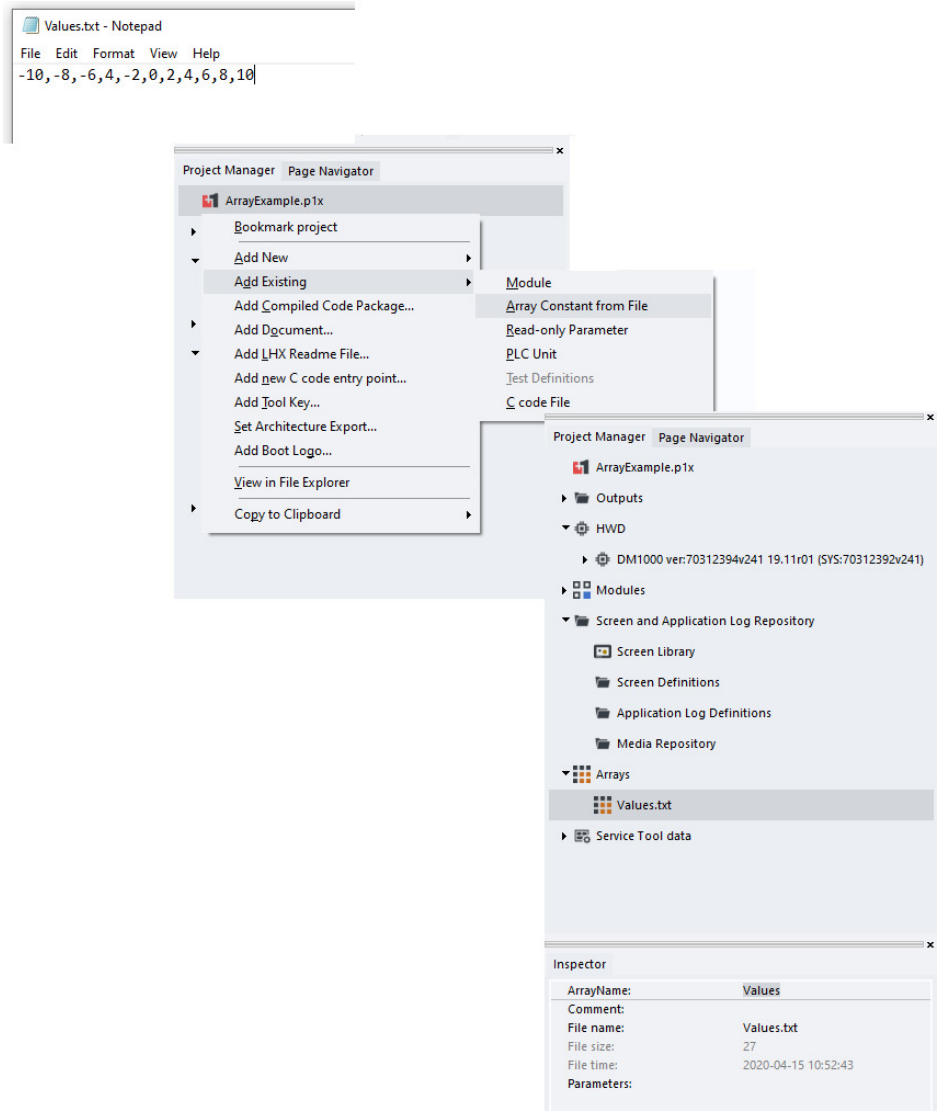


Example of resulting values 27, 51, 33 displayed when logging "CP_Values in the PLUS+1 Service Tool"

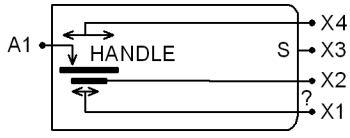


Programming

Example—Array Constant from File



Array Constant from Binary File



Use: Input data from binary (***.bin) file

Function:

- A1 = Starting position for the first element in the array
- X1 = Indicates number of valid elements in the array
- X2 = Array of data from binary file
- X3= Status

Programming

  - X3 = 0—data is ready
  - X3 ≠ 0—data is not ready
- X4 = Indicates total length of binary file
- **HANDLE** = Binary file name (You can replace **HANDLE** with a meaningful name)
  - The file name must start with a capital letter:

    OK: Binary1.bin

    Not OK: binary1.bin
  - The file name must not contain a period (.):

    OK: Binary1.bin

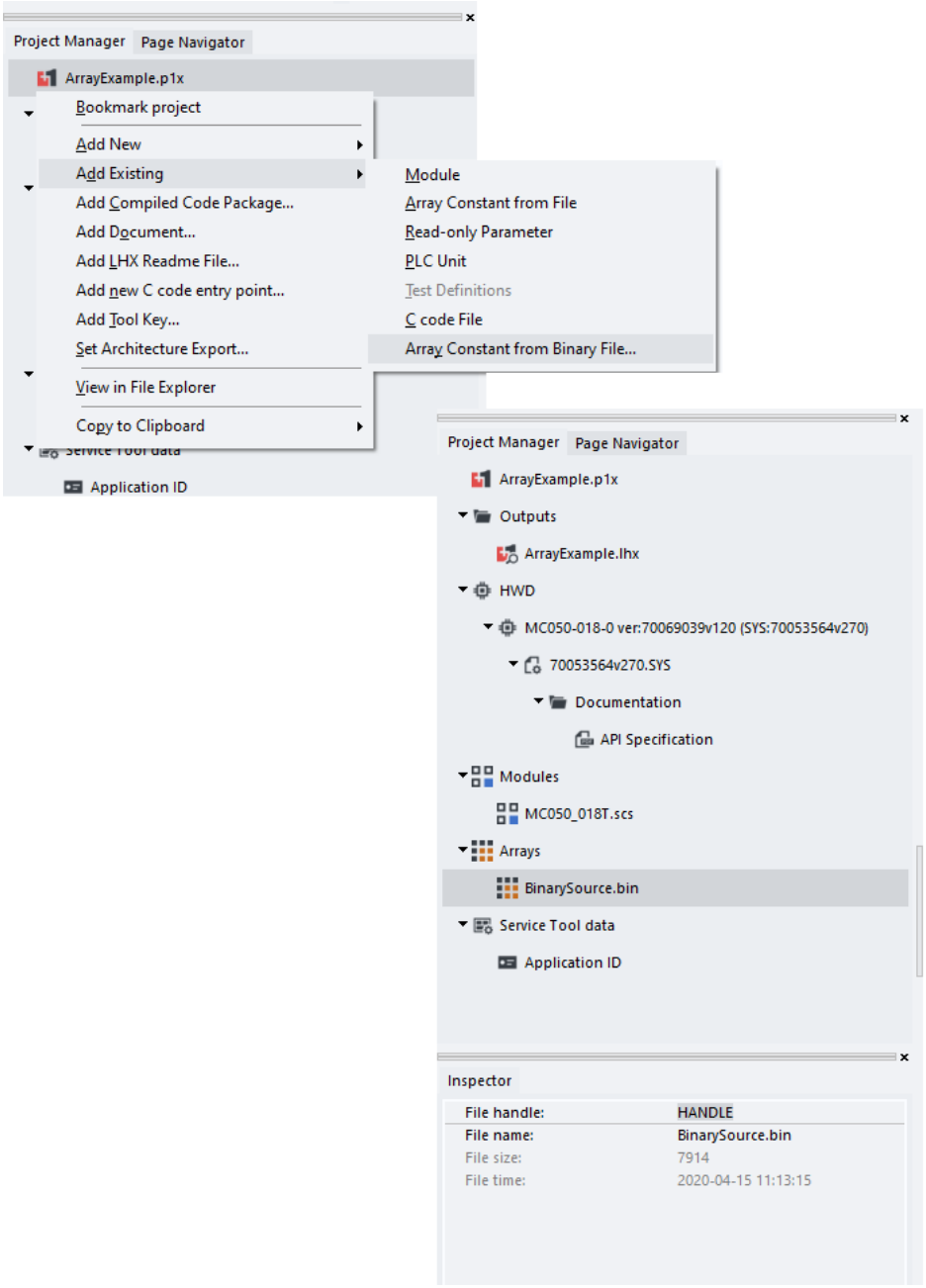
    Not OK: Binary.1.bin

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
—	——	X2	ARRAY
—	——	X3	INT
—	——	X4	INT

Programming

Example—Array Constant from Binary File



About Text and Binary Files

The **Array Constant from File** component inputs data from a text file.

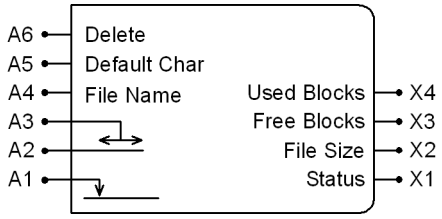
- A text file has its values in text format, delimited (separated) by commas.
- The size of an array constant input by this component is limited to about 16000 elements.

The **Array Constant from Binary File** component inputs data from a binary file.

Programming

- A binary file has its values in a sequence of bytes, with no delimiter characters...
- The size of an array constant input by this component is limited only by the device's available memory.
- Because an array constant input by this component can be very large, it may be necessary to sequentially input portions of the array.

Write Array to File



Use:

- Writes any array to an external file on a USB memory stick or other storage device
- Works with the **Read Array from Application Log** component to enable access to application data logs without the use of the PLUS+1 Service Tool program

Function:

- A1 = The position of the first byte of data to be written to the external file
  - If A1 = 0xFFFFFFFF, then new data appends to the existing data
  - The byte position count starts at zero
  - The A5 value fills any unused bytes that occur between successive writes of data
  - Ignored if X1 ≠ 0
- A2 = The array of values that write to the file when A3 ≠ 0
  - Once a write starts, the component buffers array values; array values do not update until the component finishes writing the current values
  - Ignored if X1 ≠ 0
- A3 = The number of elements in the array to write to the external file
  - If A3 = 0—do not write
  - If A3> than the array length, then the A5 value fills in after the end of the array
  - If the size of the elements in the array is larger than one byte (U8 or S8 data types), then the data writes to the file in LSB order
  - Ignored if X1 ≠ 0
- A4 = The name of the external file
  - This name must be a valid FAT 8.3 file name (an eight character filename, then a period, and a three character extension)
  - Changing a file name or attaching a storage medium causes a temporary pause as information is collected; X1 = 1 (busy) and no data can be written

    The pause does not affect the loop time; all work is done in the background
  - Ignored if X1 bit 0 ≠ 0
- A5 = Default character; fills unused bytes between successive writes of data; also fills in any unspecified elements after the array
  - Ignored if X1 ≠ 0
- A6 = True—delete the external file
  - If A6 = True and A3 ≠ 0, then the external file will first be deleted and then recreated with new data
  - Ignored if X1 ≠ 0
- X1 = Indicates the status of the component

Programming

- All X1 bits must = 0 before any inputs are sampled
- If bit:

  0 = 1—busy; checking the file name for 8.3 compliance, erasing the file, writing the file, or reading the file or the file system information

  1 = 1—current operation successfully done; the next program loop resets this bit

  2 = 1—error, work prematurely stopped; the next program loop resets this bit

  3 = 1—bad file name; the sampled file does not have a valid FAT 8.3 name; this bit must be ignored when bit 0 = 1

  4 = 1—no write medium is attached; this bit updates at least once per second
- Bits 5–31—reserved and always = 0

• X2 = Size in bytes
  - If X2 = -1—then the external file existence and size is not available because:

    Not implemented

    Busy—X1 bit 0 = 1

    Current operation is not successfully done—X1 bit 1 = 0

    The operation is successfully done (X1 bit 1 =1), but the file does not exist

    Error—X1 bit 2 = 1

    Bad file name—X1 bit 3 = 1

    No write medium attached—X1 bit 4 = 1

• X3 = Free memory (kB) in the external storage device
  - If X3 = -1, then the free memory is not available because:

    Not implemented

    Busy—X1 bit 0 = 1

    Current operation not successfully done—X1 bit 1 = 0

    Error—X1 bit 2 = 1

    Bad file name—X1 bit 3 = 1

    No write medium attached—X1 bit 4 = 1

• X4 = Used memory (kB) in the external storage device
  - If X4 = -1, then the used memory is not available because:

    Not implemented

    Busy—X1 bit 0 = 1

    Current operation is not successfully done—X1 bit 1 = 0

    Error—X1 bit 2 = 1

    Bad file name—X1 bit 3 = 1

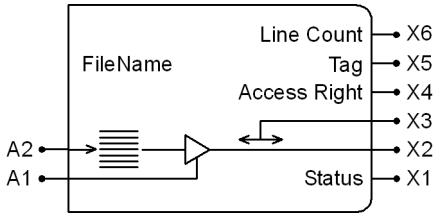
    No write medium is attached—X1 bit 4 = 1

Valid Connections

Pin	Data Type	Pin	Data Type
A1	UINT	X1	U32
A2	ARRAY[] INT	X2	S32
A3	UINT	X3	S32
A4	TL	X4	S32
A5	U8	—	——
A6	BOOL	—	——

Programming

Read Array from Application Log



Use:

- Reads and outputs an array from an application data log using any array type
- Works with the **Write Array to File** component to enable access to application data logs without the use of the PLUS+1 Service Tool program

Function:

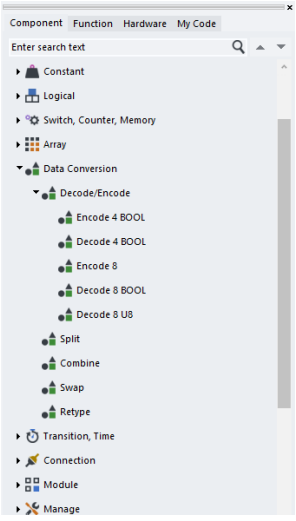
- **FileName** = Enter the file name of the application log
- A1 = True—enable read of the application log
  - The status indicated by X1 is only valid if A1 = True
- A2 = Specifies the index of the first line of data to be read from application data log
  - The line position count starts at zero
- X1 = Bits indicate status (valid only if A1 = True):
  - 00000—OK
  - 00001—data is not ready; output data values are not yet valid
  - 00010—line index overflow; A2 input points to a line beyond the last line in the application data log
  - 00100—line data overflow; data line larger than data buffer; truncated output data
  - 01000—write operation in progress
  - 10000—line count is not ready
- X2 = Read data; regardless of the data type, one application log value will be stored in one array element
- X3 = Number of bytes in the line of data being read
  - This value is not affected by the size of the output buffer
- X4 = Access right for the line of data
- X5 = Tag (data category)
  - 69 = E (Error)
  - 83 = S (Status)
  - 79 = O (Other)
- X6 = The total number of lines in the application data log

Valid connections

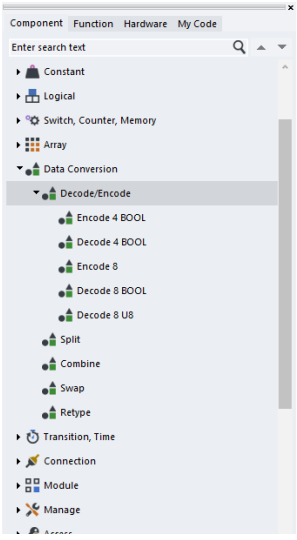
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	U32
A2	UINT	X2	ARRAY[]UINT
—	——	X3	U32
—	——	X4	UINT
—	——	X5	UINT
—	——	X6	U32

Programming

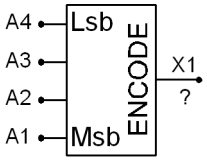
Data Conversion Menu



Decode/Encode Menu



Encode 4 BOOL



Use:

- Encodes Boolean signals into an integer signal
- Can merge Boolean data into a CAN message

Function:

- In the X1 output:

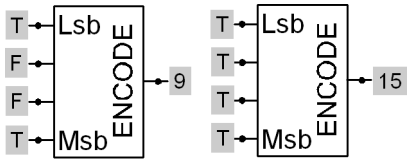
Programming

- The A1 input state sets bit 3
- The A4 input state sets bit 0

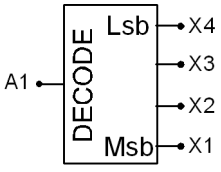
Valid connections

Pin	Data Type	Pin	Data Type
A1–A4	BOOL	X1	INT

Example—Encode 4 BOOL



Decode 4 BOOL



Use: Decodes an integer signal into four Boolean signals.

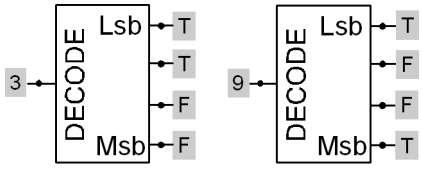
Function:

- X1 = Output of bit 3
- X4 = Output of bit 0
- A negative value on A1 sets all outputs to zero

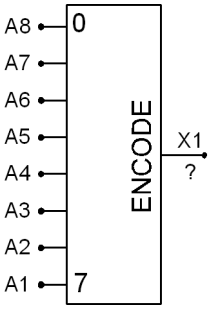
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1–X4	BOOL

Example—Decode 4 BOOL



Encode 8



Use:

Programming

- Encodes BOOL signals into an INT or ARRAY[x]BOOL
- Encodes INT signals into an ARRAY[x]INT
- Can merge data into CAN data

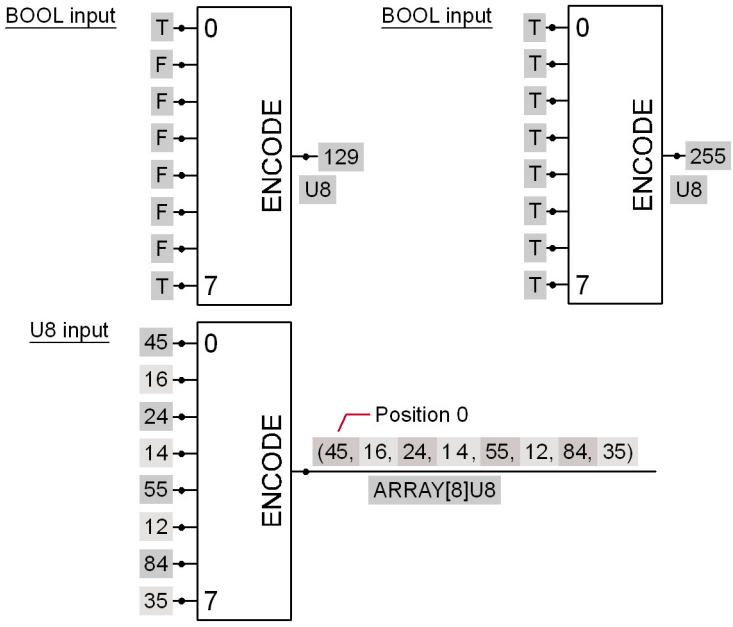
Function:

- For an INT output on X1, the:
  - A1 input sets bit 7
  - The A8 input sets bit 0
- For an array output on X1, the:
  - A1 input sets the value of the element in position 7
  - A8 input sets the value of the element in position 0

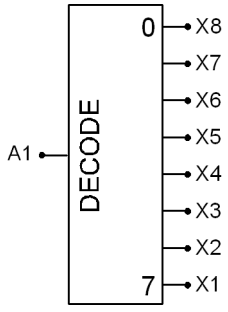
Valid connections

Pin	Data Type	Pin	Data Type
A1–A8	BOOL, INT	X1	ARRAY[BOOL], ARRAY[INT], INT

Example—Encode 8



Decode 8 BOOL



Use:

Programming

- Decodes an integer signal into eight Boolean signals
- Can extract data from a CAN data byte

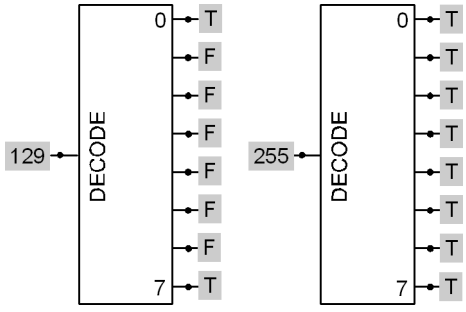
Function:

- X1 = Output of bit 7
- X8 = Output of bit 0
- A negative value on A1 sets all outputs to zero

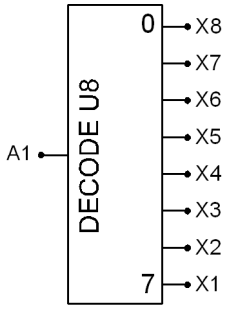
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1–X8	BOOL

Example—Decode 8 BOOL



Decode 8 U8



Use:

- Decodes an eight element array that contains integers into eight U8 signals
- Can extract data bytes from a CAN data field

Function:

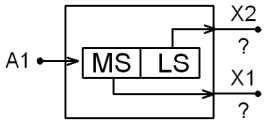
- X1 = Outputs the value of the element in position 7
- X2–X7 = Outputs the values of the elements in positions 6 through 1
- X8 = Outputs the value of the element in position 0

Valid connections

Pin	Data Type	Pin	Data Type
A1	ARRAY[8]U8	X1–X8	U8

Programming

Split



Use:

- Separates an integer signal into two signals each having an equal number of bits
- Can separate a combined CAN message into two parts

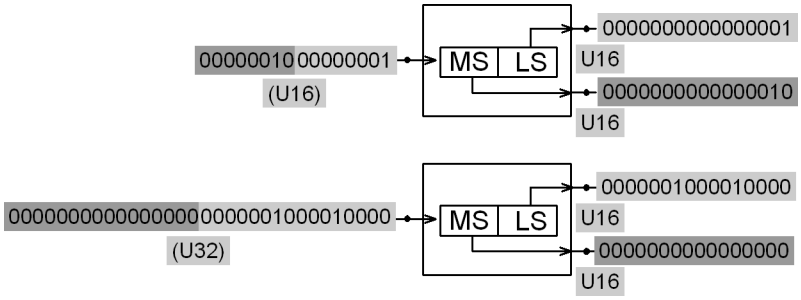
Function:

- X1 = Half of the A1 input, with the most significant A1 bits
- X2 = Half of the A1 input, with the least significant A1 bits

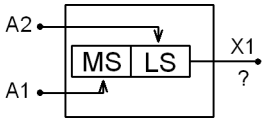
Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
—	——	X2	INT

Example—Split



Combine



Use:

- Combines two integer signals of the same size into a new signal with double the number of bits
- Can combine two separate received U8 CAN messages into a single U16 signal

Function:

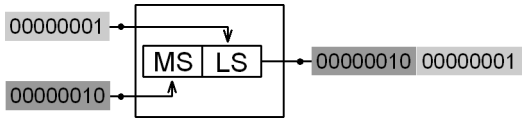
- X1 = A1 shifted to the left of A2
- Always combine inputs of the same bit size
  - An 8-bit A1 input and an 8-bit A2 input combine to produce a 16-bit X1 output
  - A 16-bit A1 input and a 16-bit A2 input combine to produce a 32-bit X1 output

Valid connections

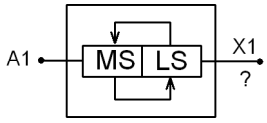
Pin	Data Type	Pin	Data Type
A1	INT	X1	INT
A2	INT	—	——

Programming

Example—Combine



Swap



Use: Swaps the bits in an integer signal.

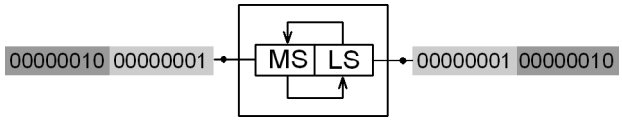
Function:

- X1 = Least significant half of the bits in the A1 input swap places with the most significant half of the bits in the A1 input
- After a swap:
  - The 8 most significant bits in a 16-bit A1 input become the 8 least significant bits in the X1 output
  - The 8 least significant bits in a 16-bit A1 input become the 8 most significant bits in an X1 output

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	X1	INT

Example—Swap



Retype

Use: Changes the data type of a signal.

Function:

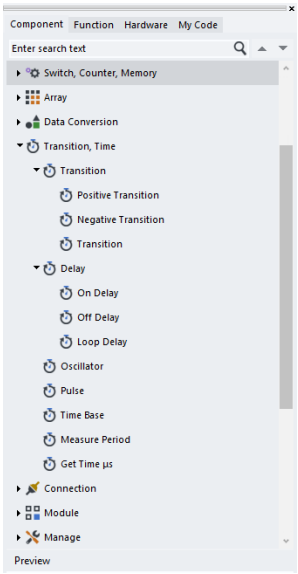
- X1 = A1
- Changing data types, such as going from S8 to U8 or from U16 to U8, can cause the X1 output to overflow

Valid connections

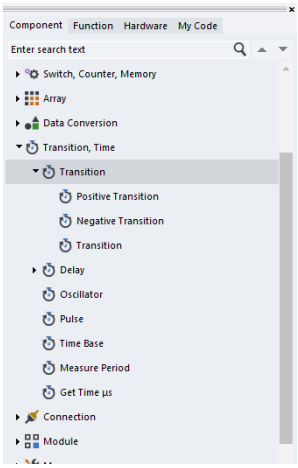
Pin	Data Type	Pin	Data Type
A1	INT, ARRAY[]U8, TL, STRING	X1	INT, ARRAY[]U8, TL, STRING

Programming

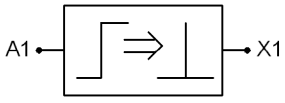
Transition, Time Menu



Transition Menu



Positive Transition



Use: Detects the positive transition (False/True) of a Boolean signal

Function: X1 = True for one program loop after A1 transitions False/True

Valid connections

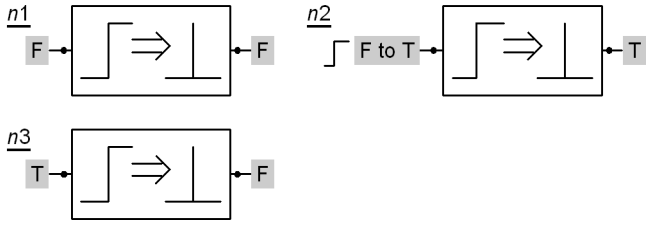
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL

Programming

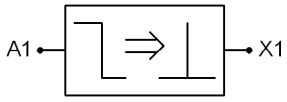
X1 Output

Program Loop	A1	X1
n1	False	False
n2	True	True
n3	True	False

Example—Positive Transition



Negative Transition



Use: Detects the negative transition (True/False) of a Boolean signal

Function: X1 = True for one program loop after A1 transitions True/False

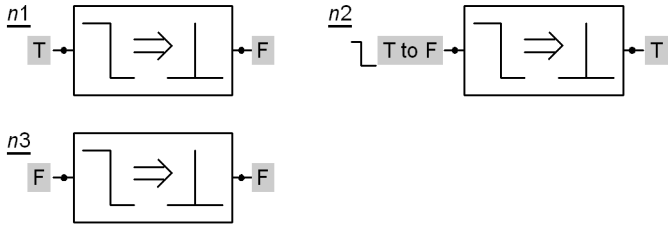
Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL

X1 Output

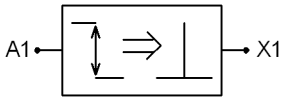
Program Loop	A1	X1
n1	True	False
n2	False	True
n3	False	False

Example—Negative Transition



Programming

Transition



Use: Detects a False/True or a True/False transition of a Boolean signal

Function: X1 = True for one program loop after A1 transitions False/True or True/False

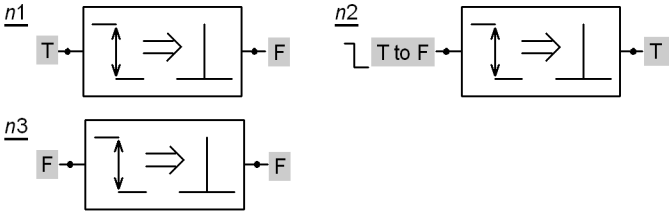
Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL

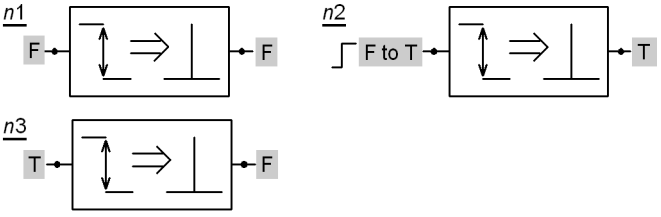
X1 Output

Example 1 Program Loop	A1	X1	Example 2 Program Loop	A1	X1
n1	True	False	n1	False	False
n2	False	True	n2	True	True
n3	False	False	n3	True	False

Example1—Transition

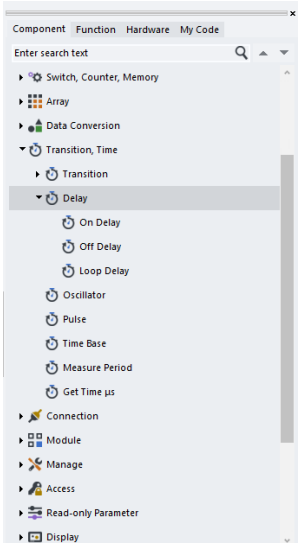


Example2—Transition

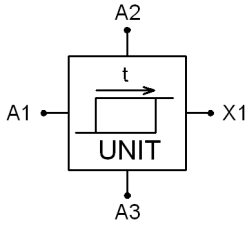


Programming

Delay Menu



On Delay



Use: Delays the return of a Boolean signal from False/True for a specified amount of time

Function:

- X1 = False if A1 = False
- When A1 goes from False/True, the hold time (A3 x A2) begins
- X1 = False during the hold time
- X1 = True when the hold time ends; X1stays True until A1 = False
- If A3 = TLOOP, then the delay time is A2 – 1

  (Example—if A3 is TLOOP and A2 is 2, then the delay time is 1 program loop)
- If the component is applying a delay time, a change in the A2 value does not change the hold time until the current delay time completes

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	UINT	—	——
A3	TIME	—	——

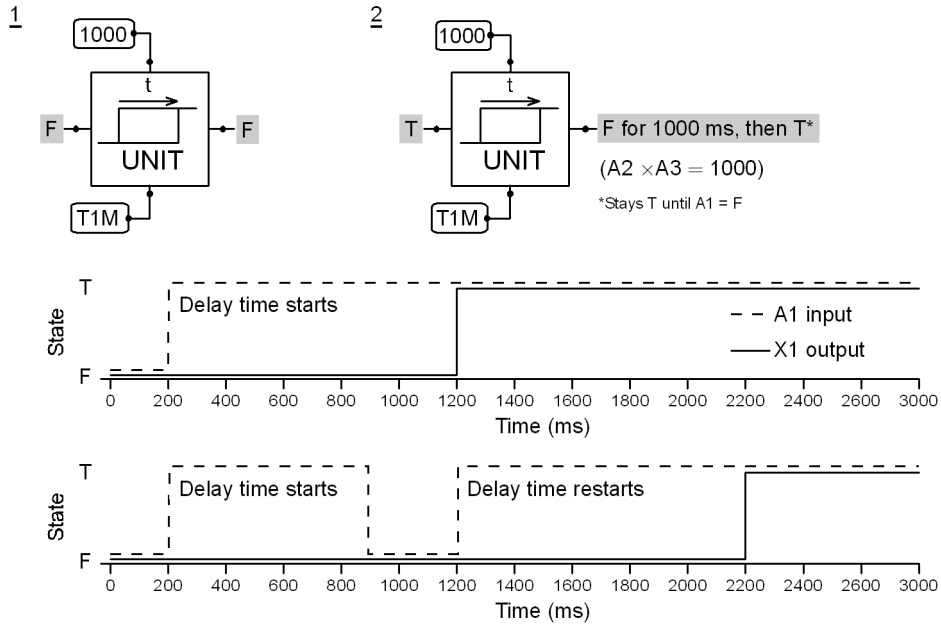
Programming

Time Units

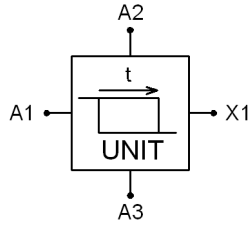
Unit	Unit Value	Unit	Unit Value	Unit	Unit Value
T1M	1 ms	T1S	1 s	TLOOP	Program loop
T10M	10 ms	T60S	60 s	—	——
T100M	100 ms	TIH	1 h	—	——

Programming

Example—On Delay



Off Delay



Use: Delays the return of a Boolean signal from True/False for a set time

Function:

- X1 = True if A1 = True
- When A1 goes from True/False, the hold time (A3 x A2) begins
- X1 = True during the hold time
- X1 = False when the hold time ends; X1stays False until A1 = True
- If A3 = TLOOP, then the delay time is A2 – 1

  (Example—if A3 is TLOOP and A2 is 2, then the delay time is 1 program loop)
- If the component is applying a delay time, a change in the A2 value does not change the hold time until the current delay time completes

Valid connections

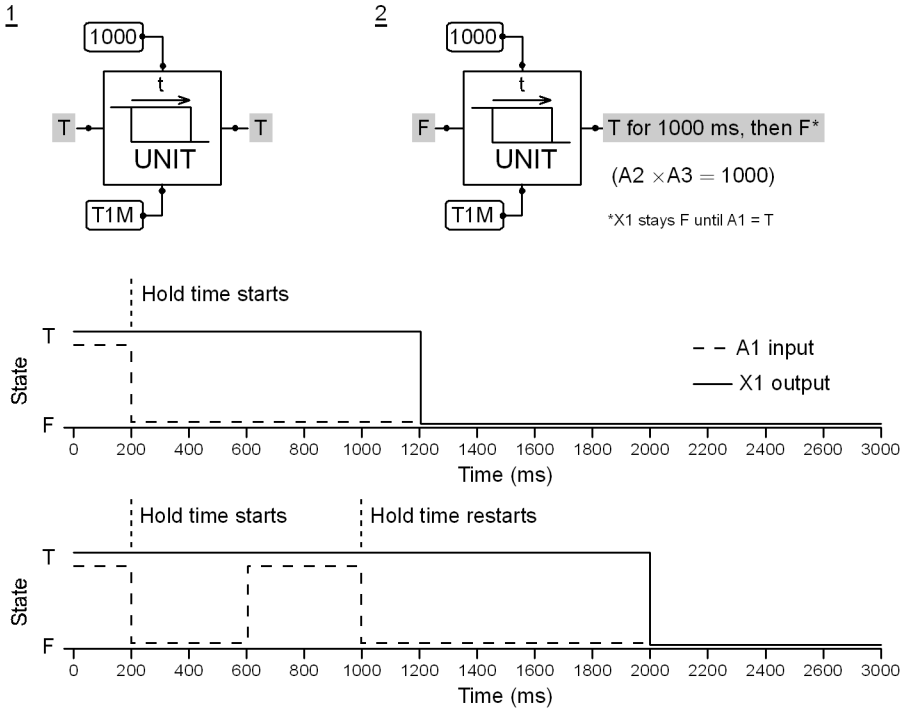
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	UINT	—	——
A3	TIME	—	——

Programming

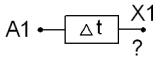
Time Units

Unit	Unit Value	Unit	Unit Value	Unit	Unit Value
T1M	1 ms	T1S	1 s	TLOOP	Program loop
T10M	10 ms	T60S	60 s	—	——
T100M	100 ms	TIH	1 h	—	——

Example—Off Delay



Loop Delay



Use: Passes the A1 input to X1 after a delay of one execution of the component

Function: X1 = A1 after a delay of one program loop

Valid connections

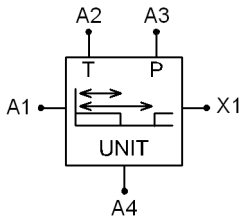
Pin	Data Type	Pin	Data Type
A1	BOOL, INT, TIME, FILE, FONT, PORT, COL, OBJ	X1	BOOL, INT, TIME, FILE, FONT, PORT, COL, OBJ

Loop Delay Component Input/Output

Program Loop	A1 Input	X1 Output
n1	5	5
n2	7	5
n3	7	7
n4	7	7

Programming

Oscillator



Use:

- Generates Boolean signals that toggle
- Combine with a counter to make a timer

Function:

- A1 is the enable input:
  - When A1 goes from False/True, a new period immediately starts
  - As long as A1 stays True, a new period starts as soon as the preceding period finishes (A2 and A3 get resampled when a new period starts)
  - When A1 goes from True/False, the current period immediately cancels
- A2 sets the on time (T in the component)
  - A2 is always (and only) sampled at the start of a new period
- A3 sets the period (P in the component):
  - A3 is always (and only) sampled at the start of a new period
- A4 determines the time unit of A2 and A3
- X1 is the output
  - X1 = False when A1 = False
  - When A1 = True, the following applies:

    X1 = True for a minimum of one loop during the first (A2) hold time, provided that A2 > 0

    X1 = False for a minimum of one loop during the second (A3 – A2) hold time, provided that A3 – A2 > 0
- Loop time limits the resolution of this component
  - This component is divided into duration for on time (A2) and duration for off time (A3 - A2), so the overall impact will be twice as big as described in *Resolution* on page 184.
- The shortest possible period time is 2 x the loop time; this applies in normal use, where A2 > 0 and A3 – A2 > 0

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	UINT	—	——
A3	UINT	—	——
A4	TIME	—	——

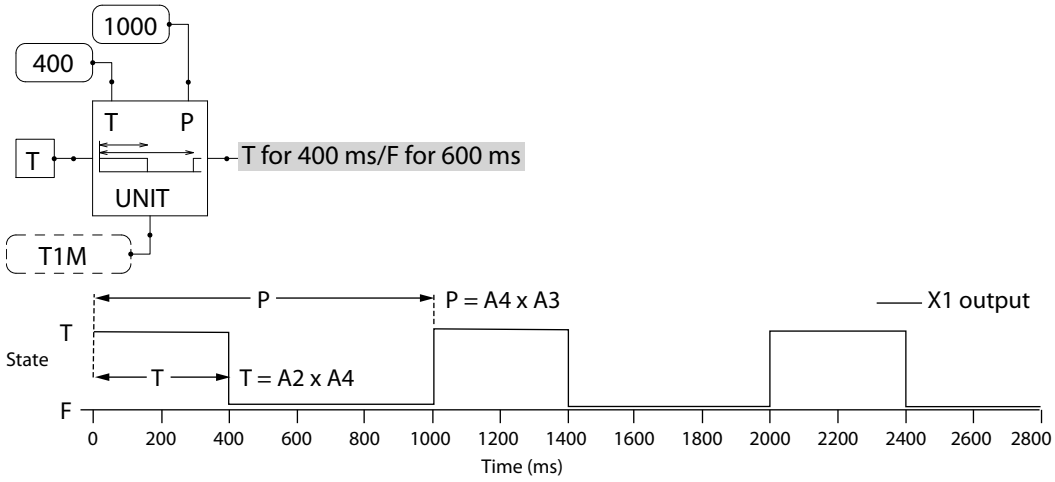
Time Units

Unit	Unit Value	Unit	Unit Value	Unit	Unit Value
T1M	1 ms	T1S	1 s	TLOOP	Program loop
T10M	10 ms	T60S	60 s	—	——
T100M	100 ms	TIH	1 h	—	——

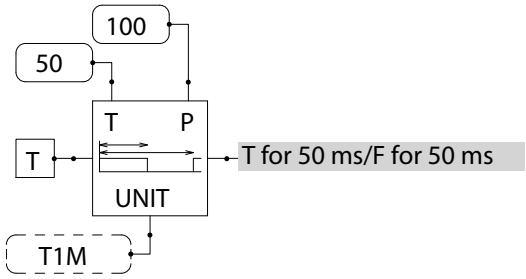
Programming

Example—Oscillator

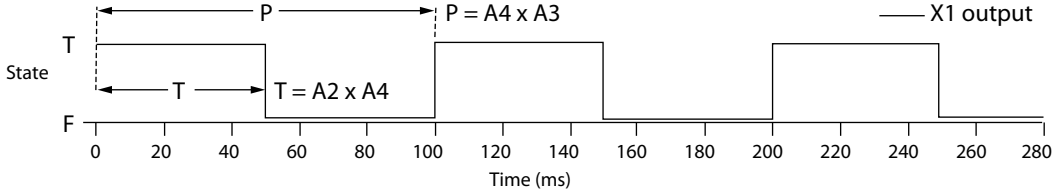
Example: Oscillator



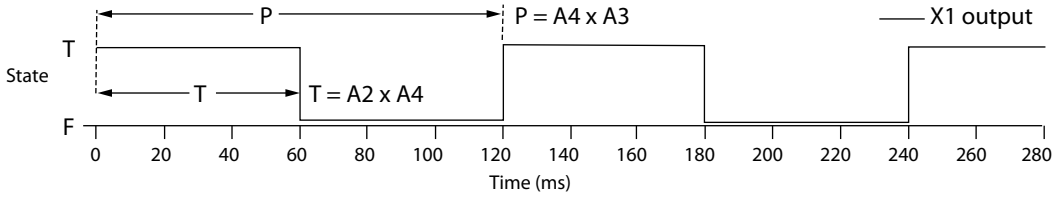
Example: Loop time impact on the result



Loop time = 1 ms: T for 50 ms/F for 50 ms

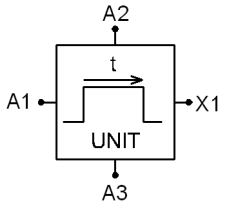


Loop time = 15 ms: T for 60 ms/F for 60 ms



Programming

Pulse



Use: Holds a Boolean signal at True for a specified time before returning the signal to False

Function:

- If A1 = False, then X1 = False
- The hold time (A3 x A2) begins when A1 goes from False/True
- X1 = True during the hold time
- If A1 = True/False during the hold time, then X1 = False
- X1 = False when the hold time ends

Valid connections

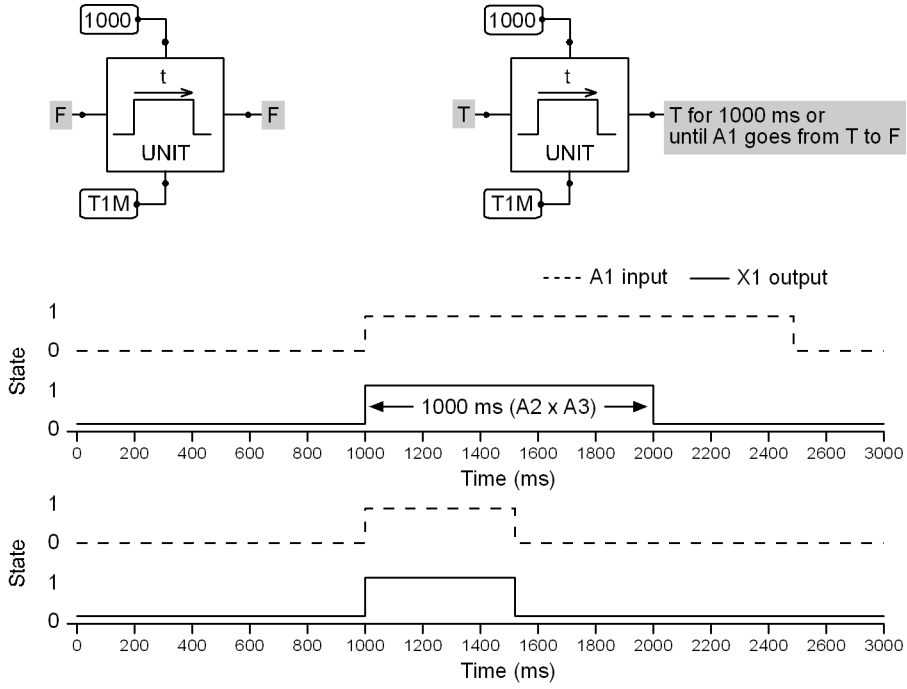
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	UINT	—	——
A3	TIME	—	——

Time Units

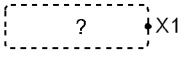
Unit	Unit Value	Unit	Unit Value	Unit	Unit Value
T1M	1 ms	T1S	1 s	TLOOP	Program loop
T10M	10 ms	T60S	60 s	—	——
T100M	100 ms	TIH	1 h	—	——

Programming

Example—Pulse



Time Base



Use: Outputs a constant time base.

Function: X1 = Time base value that replaces **?**.

Valid connections

Pin	Data Type	Pin	Data Type
—	—	X1	TIME

Time Units

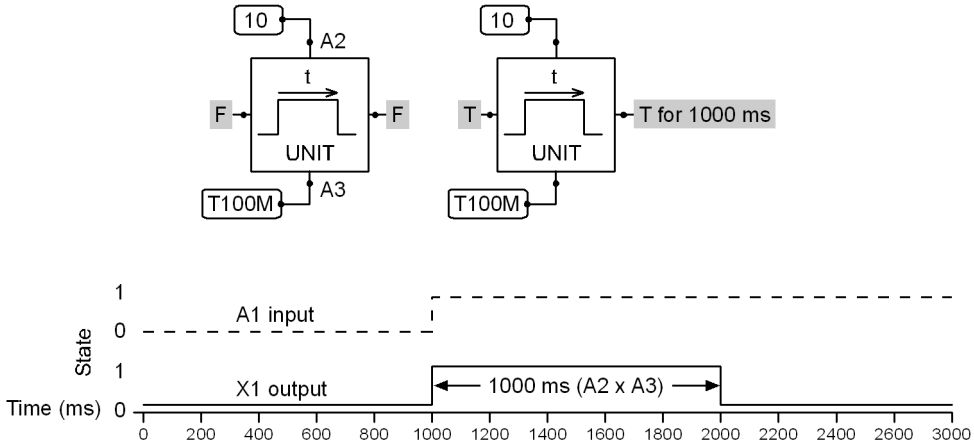
Unit	Unit Value	Unit	Unit Value	Unit	Unit Value
T1M	1 ms	T1S	1 s	TLOOP	Program loop
T10M	10 ms	T60S	60 s	—	——
T100M	100 ms	TIH	1 h	—	——

A smaller value time base produces a more accurate output resolution.

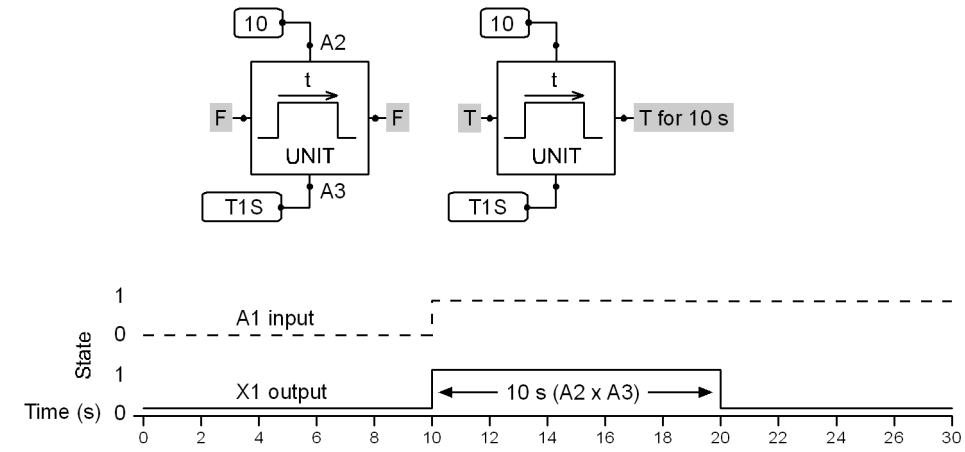
Programming

Example—Time Base

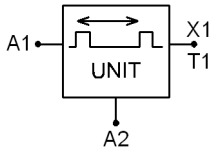
Pulse component with a 100 ms Time Base



Pulse component with a 1 s Time Base



Measure Period



Use: Measures the period of a toggling signal

Function:

- X1 = Time between the two last False/True transitions on A1
- X1 = A1 ÷ A2

Valid connections

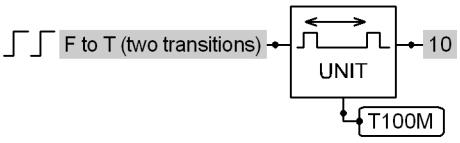
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	UINT
A2	TIME	—	——

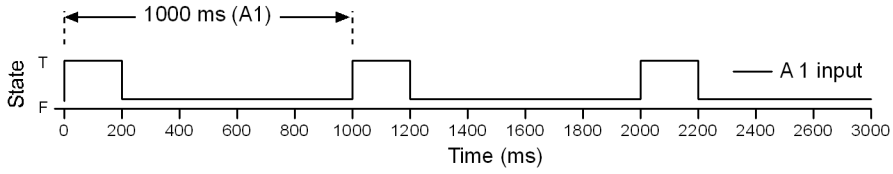
Programming

Time Units

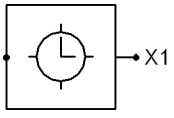
Unit	Unit Value	Unit	Unit Value	Unit	Unit Value
T1M	1 ms	T1S	1 s	TLOOP	Program loop
T10M	10 ms	T60S	60 s	—	——
T100M	100 ms	TIH	1 h	—	——

Example—Measure Period





Get Time µs



Use:

- Outputs OS.LoopCnt time in microseconds (µs)
- Measure the execution time of logic
- Hardware determines resolution

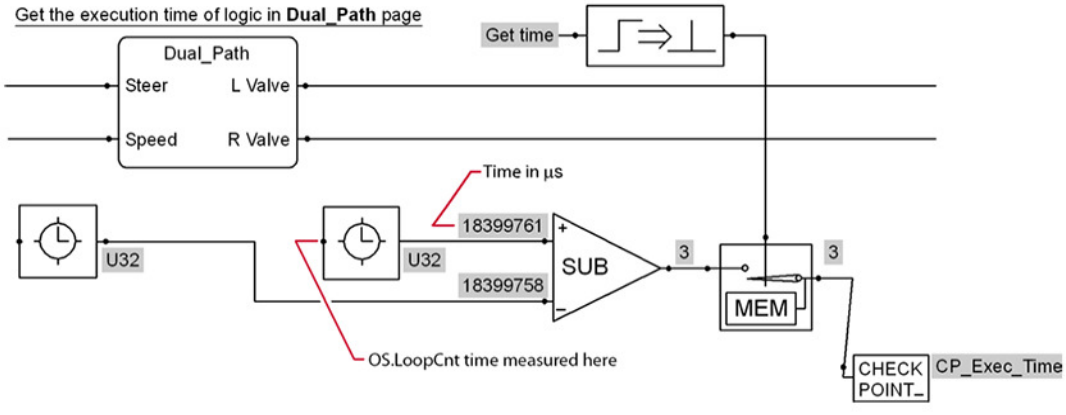
Function: X1 = Execution time of component in µs

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	U32

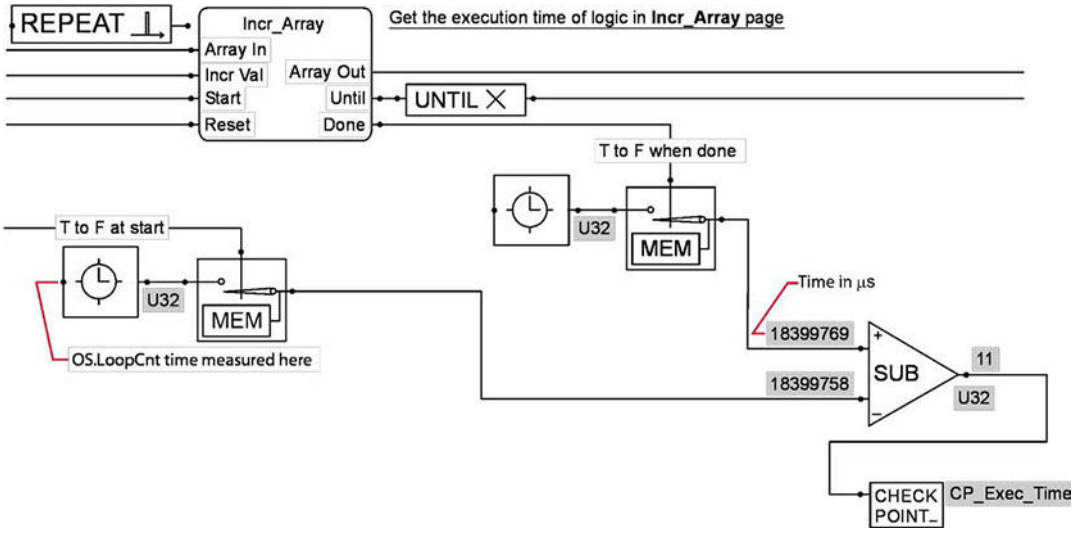
The order of PLUS elements (components and pages) affects the execution time. See *About Execution Order* on page 187.

Example 1—Get Time µs



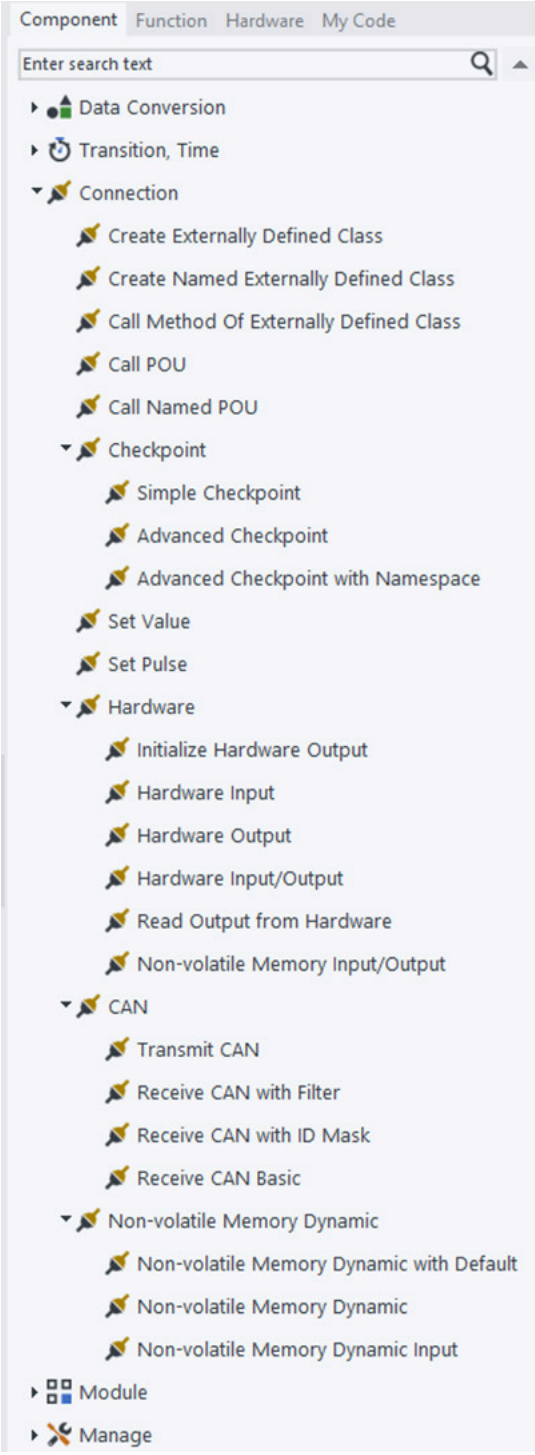
Programming

Example 2—Get Time µs

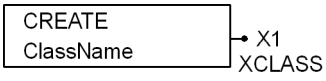


Programming

Connection Menu



Create Externally Defined Class



Use:

Programming

- Includes compiled C code contained in a compiled code package (CCP)
- Creates a C code class instance that contains methods that may be called
- May be used with one or more **Call Method of Externally Defined Class** components (this component calls a method of the C code class)
- If no **Call Method of Externally Defined Class** component is connected, then using the **Create Externally Defined Class** component adds no functionality

Function:

- **ClassName** = Enter the name of the C code class
- X1 = Creates an instance of a C code class contained in a CCP

  Connect this output to the **INSTANCE** input on the **Call Method of Externally Defined Class** component
- No code executes from where the **Create Externally Defined Class** component is placed

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	XCLASS

For more information on how to create the interface between a compiled code package and the PLUS+1® application, see *PLUS+1® GUIDE Create Compiled Code Package, User Manual*, AQ152886480347.

Reference *Example—Create Externally Defined Class/Call Method of Externally Defined Class* on page 306.

Warning

Using an untested compiled code package (CCP) in an application can produce an application that is unstable and unpredictable. An unstable, unpredictable application can cause unexpected machine movements that result in personal injury and equipment damage. To reduce the risks in using an application that contains a CCP, both the creator and the user of the application must first be certain that the CCP:

- Contains only code that was compiled using the compiler and the settings recommended by Danfoss.
- Operates with its intended hardware correctly and without defects.
- Does not contain code with lengthy execution times that can cause the controller to significantly slow down or freeze.
- Does not contain code with potential endless loops and “wait until” statements.
- Follows the CCP guidelines recommended by Danfoss.
- Passes data only through the bus structure recommended by Danfoss.
- Does not access the hardware application programming interface or any other platform resources such as the RAM, EEPROM, CPU registers, and kernel.
- Accesses only the memory allocated to the compiled code package.
- Keeps the use of stack memory to a minimum.

  In addition to testing the CCP, the creator of an application that contains a CCP, must thoroughly test the application itself.

Create Named Externally Defined Class

Use:

Programming

- Includes compiled C code contained in a compiled code package (CCP)
- Creates a C code class instance that contains methods that may be called
- May be used with one or more **Call Method of Externally Defined Class** components (this component calls a method of the C code class)
- If no **Call Method of Externally Defined Class** component is connected, then using the **Create Named Externally Defined Class** component adds no functionality
- Should be used instead of the **Create Externally Defined Class** component when you want to include Diagnostic and Non-volatile parameters of the CCP in the diagnostic data.

Function:

- **InstanceName** = Enter the name of the instance
- **ClassName** = Enter the name of the C code class
- X1 = Creates an instance of a C code class contained in a CCP.

  Connect this output to the **INSTANCE** input on the **Call Method of Externally Defined Class** component
- No code executes from where the **Create Named Externally Defined Class** component is placed

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	XCLASS

For more information on how to create the interface between a compiled code package and the PLUS+1® application, see *PLUS+1® GUIDE Create Compiled Code Package, User Manual*, AQ152886480347.

Reference *Example—Create Externally Defined Class/Call Method of Externally Defined Class* on page 306.

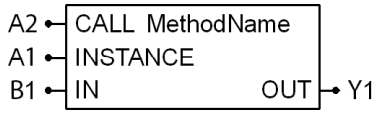
Warning

Using an untested compiled code package (CCP) in an application can produce an application that is unstable and unpredictable. An unstable, unpredictable application can cause unexpected machine movements that result in personal injury and equipment damage. To reduce the risks in using an application that contains a CCP, both the creator and the user of the application must first be certain that the CCP:

- Contains only code that was compiled using the compiler and the settings recommended by Danfoss.
- Operates with its intended hardware correctly and without defects.
- Does not contain code with lengthy execution times that can cause the controller to significantly slow down or freeze.
- Does not contain code with potential endless loops and “wait until” statements.
- Follows the CCP guidelines recommended by Danfoss.
- Passes data only through the bus structure recommended by Danfoss.
- Does not access the hardware application programming interface or any other platform resources such as the RAM, EEPROM, CPU registers, and kernel.
- Accesses only the memory allocated to the compiled code package.
- Keeps the use of stack memory to a minimum.

  In addition to testing the CCP, the creator of an application that contains a CCP, must thoroughly test the application itself.

Call Method of Externally Defined Class



Use:

Programming

- Defines a point in the execution of the application where a method call is made on an object of an externally defined class type
- Must be used with one of the components **Create Externally Defined Class** or **Create Named Externally Defined Class** (those components define the C code class to which the method belongs)
- When using this component together with the **Create Named Externally Defined Class** component, there is an optional BOOL input named **#DisableCheckpoints** on the IN bus. If set to True, checkpoints of the CCP instance will not be added to the diagnostic data.

Function:

- A2 = If True, then call the function defined by the **MethodName**
- **MethodName** = Enter the function to be called from the C code class
- A1 = Connect to the **Create Externally Defined Class** or **Create Named Externally Defined Class** component's X1 output
- B1 = Inputs to the function
- Y1= Outputs from the function

Valid communications

Pin	Data type
A2	BOOL
A1	XCLASS
B1	EXTENDED
Y1	EXTENDED

Reference *Example—Create Externally Defined Class/Call Method of Externally Defined Class* on page 306.

When a method call is not made (A3 = False), the values that the method outputs remain unchanged from the previous program loop or False if no call has been made. Keep this in mind if your function controls outputs.

Warning

Using an untested compiled code package (CCP) in an application can produce an application that is unstable and unpredictable. An unstable, unpredictable application can cause unexpected machine movements that result in personal injury and equipment damage. To reduce the risks in using an application that contains a CCP, both the creator and the user of the application must first be certain that the CCP:

- Contains only code that was compiled using the compiler and the settings recommended by Danfoss.
- Operates with its intended hardware correctly and without defects.
- Does not contain code with lengthy execution times that can cause the controller to significantly slow down or freeze.
- Does not contain code with potential endless loops and "wait until" statements.
- Follows the CCP guidelines recommended by Danfoss.
- Passes data only through the bus structure recommended by Danfoss.
- Does not access the hardware application programming interface or any other platform resources such as the RAM, EEPROM, CPU registers, and kernel.
- Accesses only the memory allocated to the compiled code package.
- Keeps the use of stack memory to a minimum.

In addition to testing the CCP, the creator of an application that contains a CCP, must thoroughly test the application itself.

Programming

Warning

There is an optional BOOL input named **#Unbuffered** on the IN bus that can set the Method to run unbuffered. This input only applies to array data. In most cases it is recommended to NOT connect this input unless you are certain that the CCP code does not store pointers into GUIDE code data structures.

- **#Unbuffered** input = T (constant TRUE)

  The Method runs unbuffered. This setting is faster and uses less memory but comes with a risk that C code that is not written according to recommendations keeps pointers to arrays beyond their lifetime.
- **#Unbuffered** input = F (constant FALSE)

  The Method runs buffered even in cases where the Method itself specifies that no buffer is needed. Running buffered is a safe choice but can use a bit more memory and execution time.
- **#Unbuffered** input connected to something other than constant TRUE or FALSE (for example a BOOL variable, or even just an unconnected green wire)

  This is not how the **#Unbuffered** input should be used. The CCP Method runs buffered except in cases where the CCP Method itself specifies that it is safe to run unbuffered.
- No **#Unbuffered** input.

  This is the recommended use case. The CCP Method itself will decide specifically which inputs/outputs will be buffered and which will not be buffered.

Example—Create Externally Defined Class/Call Method of Externally Defined Class

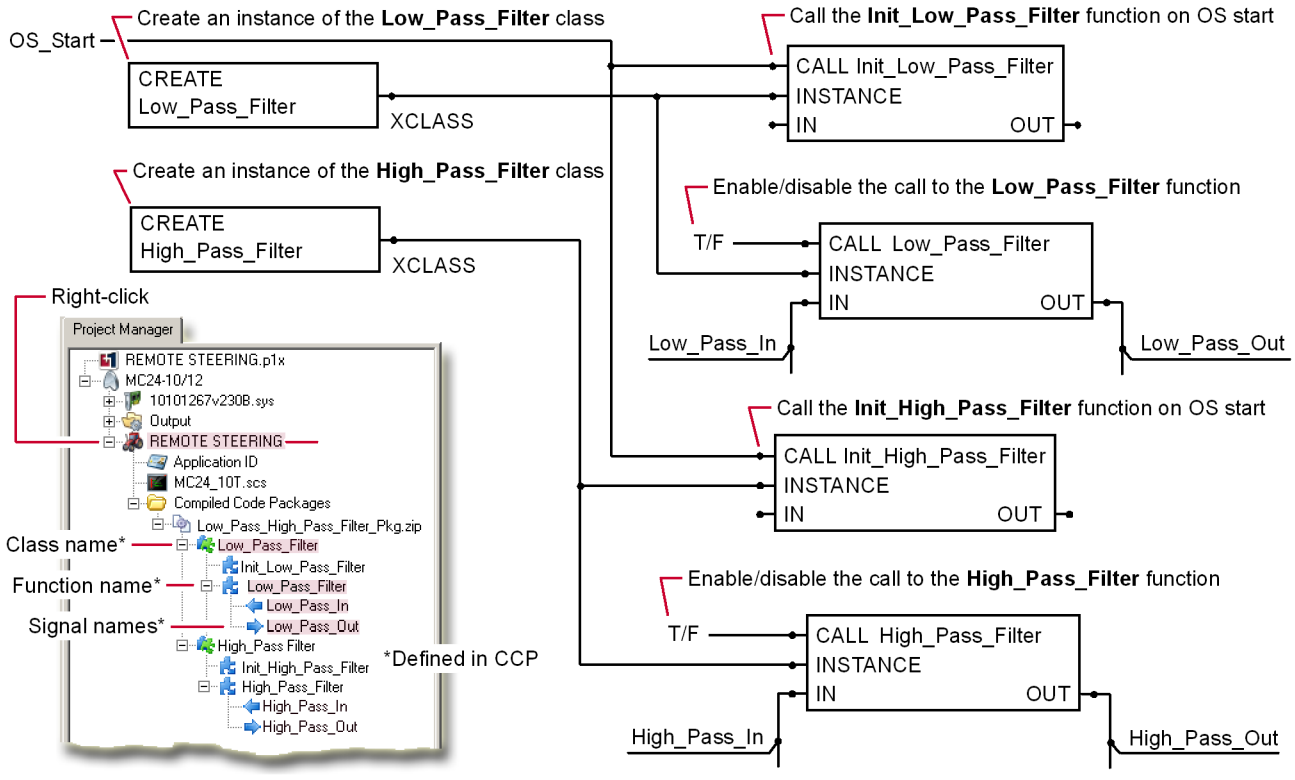
The following example shows a compiled code package (CCP) named **Low_Pass_High_Pass_Filter_Pkg** that is installed in a PLUS+1 application. The CCP contains C code that implements a low-pass filter and a high-pass filter. This C code has a **Low_Pass_Filter** class and a **High_Pass_Filter** class.

- The **Low_Pass_Filter** class has two functions:
  - An **Init_Low_Pass_Filter** function that gets called when the **OS_Signal** = True. This function initializes the low-pass filter.
  - A **Low_Pass_Filter** function that, when called, applies low-pass filtering.
- The **High_Pass_Filter** class has two functions:

Programming

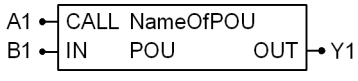
- An **Init_High_Pass_** > **Filter** function that gets called when the **OS_Signal** = True. This function initializes the high-pass filter.
- A **High_Pass_Filter** function that, when called, applies high-pass filtering.

Example—Create Externally Defined Class/Call Method of Externally Defined Class



(The names of the signals connected to the **Call Method** component must match the signals names shown in the **Inspector** panel.)

Call POU



Use:

Defines a point in the execution of an application where a call is made on a POU (Program Organization Unit).

PLUS+1® GUIDE supports the "strict mode" and the "non-strict mode" for PLC code.

- Strict mode is mandatory for any SIL2 compile. Danfoss recommends that you use the strict mode. In this mode, the PLC standard (IEC 61131-3) is followed strictly, and each use of non-standard syntax generates an error message on compile.
- Non-strict mode might be needed for importing code written using other tools. Danfoss recommends that you do not use the non-strict mode. In this mode, each use of supported non-standard syntax generates a warning message on compile.

All POUs must conform to the PLC standards as defined in IEC 61131-3.

Refer to this standard for information on how to create PLC code.

Function:

Programming

- A1 = If True, then call the POU defined by the **NameOfPOU**
- **NameOfPOU** = Enter the name of the POU to be called
- B1 = Inputs to the POU
- Y1 = Outputs from the POU

When a POU call ends (A1 = F), the values that the function outputs when the call ends continue to be output. If you control outputs with your function, you must create logic to turn these outputs off before the call ends.

Valid connections

Pin	Data Type
A1	BOOL
B1	EXTENDED
Y1	EXTENDED

Warning

Using untested PLC code in an application can produce an application that is unstable and unpredictable. An unstable, unpredictable application can cause unexpected machine movements that result in personal injury and equipment damage. To reduce the risks in using an application that contains PLC code, the creator of the application must be certain that the PLC code:

- Avoids the use of recursion, which can cause excessive use of stack memory.
- Does not contain code with lengthy execution times that can cause a controller to significantly slow down or freeze.
- Does not contain code with potential “while” and “repeat until” endless loops.
- Does not contain code with “for” loops that do not modify the loop variable, potentially causing endless loops.
- Does not contain code with backward jump instructions, potentially causing endless loops.
- Keeps the use of stack memory to a minimum. Exhausted stack memory causes undefined ECU behavior.

Warning

The creator of an application that contains PLC code should be aware of some issues that are related to floating point use:

- MC0xx-0xx units do not fully support the LREAL (F64) data type.
- It is a mistake to assume that floating point data types can precisely represent all real numbers; floating point data types have a finite precision.

  (For example, there is no exact representation of the number 0.1.)
- It is a mistake to assume that floating point data types can precisely represent all real numbers; floating point data types have a finite precision.

  (For example, there is no exact representation of the number 0.1.)
- Two calculations that mathematically should be equal might not yield equal floating point values.

  (For example, 0.1 * 0.1 is not exactly equal to 0.01 when using floating point data types.)
- Checking that the divisor is not zero does not guarantee that a division will not overflow.
- Type conversion from a floating point type to an integer type will truncate the value rather than round the value.

The developer must ensure that imported code works correctly on the intended hardware platform.

Programming

Warning

There is an optional BOOL input named **#Unbuffered** on the IN bus that can set the POU to run unbuffered. This input only applies to array data. In most cases it is recommended to NOT connect this input unless you either don't use C code POUs or are certain that the C code POUs do not store pointers into GUIDE code data structures.

- **#Unbuffered** input = T (constant TRUE)

  The POU runs unbuffered. This setting is faster and uses less memory but comes with a risk that C code that is not written according to recommendations keeps pointers to arrays beyond their lifetime. However, if the called POU is not written in C, and does not call any POU written in C, then it is safe to set #Unbuffered to TRUE.
- **#Unbuffered** input = F (constant FALSE)

  The POU runs buffered even in cases where the POU itself specifies that no buffer is needed. Running buffered is a safe choice but can use a bit more memory and execution time.
- **#Unbuffered** input connected to something other than constant TRUE or FALSE (for example a BOOL variable, or even just an unconnected green wire).

  This is not how the **#Unbuffered** input should be used.

  The POU runs buffered except in cases where the POU itself specifies that it is safe to run unbuffered.
- No **#Unbuffered** input.

  This is the recommended use case.

  The POU itself will decide specifically which inputs/outputs will be buffered and which will not be buffered.

About Pointers

PLUS+1® GUIDE release 6.1 supports the use of pointers in PLC code.

This manual does not discuss pointers as a standard programming concept. This manual only covers the details that apply to pointers as implemented for PLC in the PLUS+1® GUIDE.

Pointers are not part of the PLC standard and are not allowed in the strict mode.

Pointers in the PLUS+1® GUIDE are similar in function to the function of pointers in in Pascal. Note the following details:

- Pointer variables are declared as "**POINTER to X**", where X is a standard Structured Text data type.
- Pointer variables can be assigned to and from any other POINTER, DWORD, DINT or UDINT variable.
- The function "**adr()**" can be used to get the address of a variable.

  For example, "adr(X)" returns the address of X, where X is a variable.
- The operator "**^**" can be used to de-reference a pointer variable.

  For example, "X^" returns the data pointed to by X, where X is a pointer.

Pointer Recommendations

The following recommendations apply to the use of pointers:

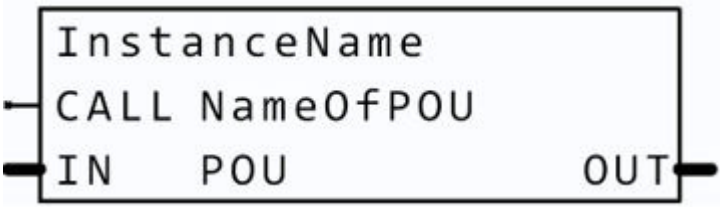
- If possible, do not use pointers.
- When using pointers, keep the following in mind:

Programming

- If possible, do not write new code using pointers; instead only use pointers where they are needed in existing code that has been imported from other tools.
- If possible, avoid any type-casting of pointers.
- If possible, avoid pointer arithmetics.

  When using pointer arithmetics, never increment by a value that is not a multiple of sizeof(X), where X is the type pointed to by the pointer being incremented.

Call Named POU



Use:

Defines a point in the execution of an application where a call is made on a POU (Program Organization Unit).

PLUS+1® GUIDE supports the “strict mode” and the “non-strict mode” for PLC code.

- Should be used instead of the **Call POU** component when you want to include Diagnostic and Non-volatile parameters of the POU in the diagnostic data.
- There is an optional BOOL input named **#DisableCheckpoints** on the IN bus. If set to True, checkpoints of the POU instance will not be added to the diagnostic data.
- Strict mode is mandatory for any SIL2 compile. Danfoss recommends that you use the strict mode. In this mode, the PLC standard (IEC 61131-3) is followed strictly, and each use of non-standard syntax generates an error message on compile.
- Non-strict mode might be needed for importing code written using other tools. Danfoss recommends that you do not use the non-strict mode. In this mode, each use of supported non-standard syntax generates a warning message on compile.

All POUs must conform to the PLC standards as defined in IEC 61131-3.

Refer to this standard for information on how to create PLC code.

Function:

- **InstanceName** = Enter the name of the instance
- A1 = If True, then call the POU defined by the **NameOfPOU**
- **NameOfPOU** = Enter the name of the POU to be called
- B1 = Inputs to the POU
- Y1 = Outputs from the POU

When a POU call ends (A1 = F), the values that the function outputs when the call ends continue to be output. If you control outputs with your function, you must create logic to turn these outputs off before the call ends.

Programming

Valid connections

Pin	Data Type
A1	BOOL
B1	EXTENDED
Y1	EXTENDED

Warning

Using untested PLC code in an application can produce an application that is unstable and unpredictable. An unstable, unpredictable application can cause unexpected machine movements that result in personal injury and equipment damage. To reduce the risks in using an application that contains PLC code, the creator of the application must be certain that the PLC code:

- Avoids the use of recursion, which can cause excessive use of stack memory.
- Does not contain code with lengthy execution times that can cause a controller to significantly slow down or freeze.
- Does not contain code with potential “while” and “repeat until” endless loops.
- Does not contain code with “for” loops that do not modify the loop variable, potentially causing endless loops.
- Does not contain code with backward jump instructions, potentially causing endless loops.
- Keeps the use of stack memory to a minimum. Exhausted stack memory causes undefined ECU behavior.

Warning

The creator of an application that contains PLC code should be aware of some issues that are related to floating point use:

- MC0xx-0xx units do not fully support the LREAL (F64) data type.
- It is a mistake to assume that floating point data types can precisely represent all real numbers; floating point data types have a finite precision.

  (For example, there is no exact representation of the number 0.1.)
- It is a mistake to assume that floating point data types can precisely represent all real numbers; floating point data types have a finite precision.

  (For example, there is no exact representation of the number 0.1.)
- Two calculations that mathematically should be equal might not yield equal floating point values.

  (For example, 0.1 * 0.1 is not exactly equal to 0.01 when using floating point data types.)
- Checking that the divisor is not zero does not guarantee that a division will not overflow.
- Type conversion from a floating point type to an integer type will truncate the value rather than round the value.

The developer must ensure that imported code works correctly on the intended hardware platform.

Programming

Warning

There is an optional BOOL input named **#Unbuffered** on the IN bus that can set the POU to run unbuffered. This input only applies to array data. In most cases it is recommended to NOT connect this input unless you either don't use C code POUs or are certain that the C code POUs do not store pointers into GUIDE code data structures.

- **#Unbuffered** input = T (constant TRUE)

  The POU runs unbuffered. This setting is faster and uses less memory but comes with a risk that C code that is not written according to recommendations keeps pointers to arrays beyond their lifetime. However, if the called POU is not written in C, and does not call any POU written in C, then it is safe to set #Unbuffered to TRUE.
- **#Unbuffered** input = F (constant FALSE)

  The POU runs buffered even in cases where the POU itself specifies that no buffer is needed. Running buffered is a safe choice but can use a bit more memory and execution time.
- **#Unbuffered** input connected to something other than constant TRUE or FALSE (for example a BOOL variable, or even just an unconnected green wire).

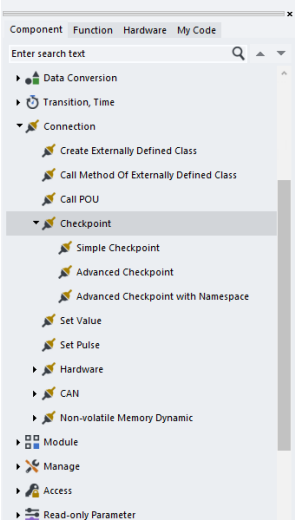
  This is not how the **#Unbuffered** input should be used.

  The POU runs buffered except in cases where the POU itself specifies that it is safe to run unbuffered.
- No **#Unbuffered** input.

  This is the recommended use case.

  The POU itself will decide specifically which inputs/outputs will be buffered and which will not be buffered.

Checkpoint Menu



Simple Checkpoint

Use:

- Creates a checkpoint signal that you can check with the PLUS+1 Service Tool program
- Route the signal that you want to check to A1; your **?** entry is the signal name
- Click the underscore (_) with the Query/Change tool to set a 0–9 PLUS+1 Service Tool program access level

The **?** names used in checkpoint and non-volatile dynamic memory components must be different. Identical **?** names produce a MULTI DEFINED SYMBOL compile error.

Programming

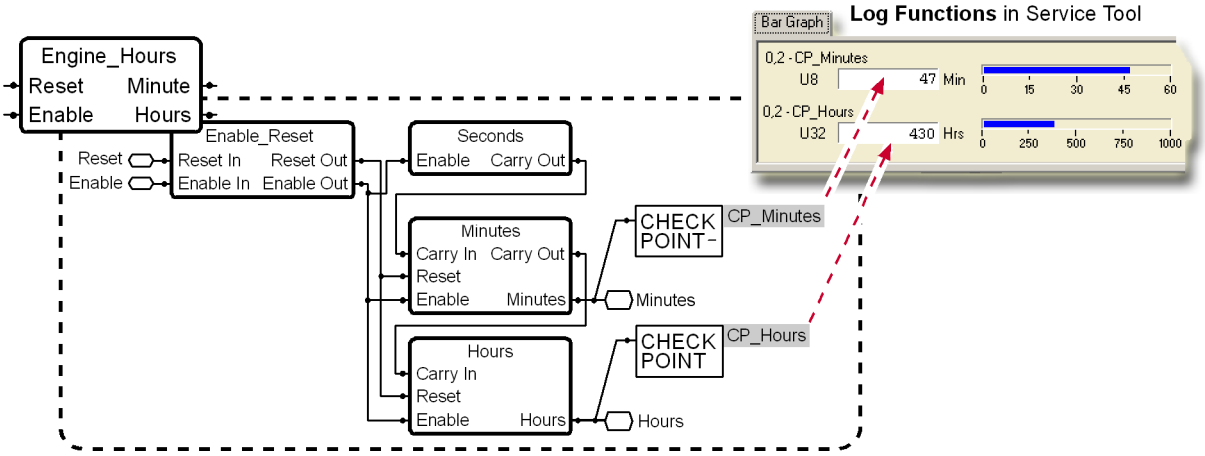
Function: **?** = User-defined name for the A1 signal

Valid Connections

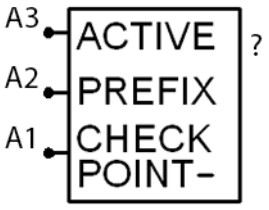
Pin	Data Type	Pin	Data Type
A1	EXTENDED	—	——

Programming

Example—Simple Checkpoint



Advanced Checkpoint



Use:

- Creates a checkpoint signal that you can check with the PLUS+1® Service Tool program
- Route the signal that you want to check to A1; your **?** entry is the signal name
- Adds the A2 PREFIX to the **?? signal name. When copying a page, you only have to change the A2 PREFIX. You do not have to change** names to use the copied page
- Click the underscore (_) with the Query/Change tool to set a 0–9 PLUS+1® Service Tool program access level
- The input to A3 must only be from a **True** or a **False** component; switched True and False inputs cause compiler errors
- The compile process makes this component permanently:
  - Active if a **True** component is connected to its A3 input
  - Inactive if a **False** signal name. When copying a page, component is connected to its A3 input
- Cannot be switched between active and inactive states by the PLUS+1® Service Tool program or through an application

The **?** names used in checkpoint and non-volatile dynamic memory components must be different. Identical **?** names produce a MULTI DEFINED SYMBOL compile error.

Function:

- **?** = User-defined name for the A1 signal
- If:

Programming

— A3 = True, then the **?** signal is available through the PLUS+1® Service Tool program

  In the PLUS+1® Service Tool program, the **?** signal name becomes the A2 PREFIX + underscore + **?** signal name

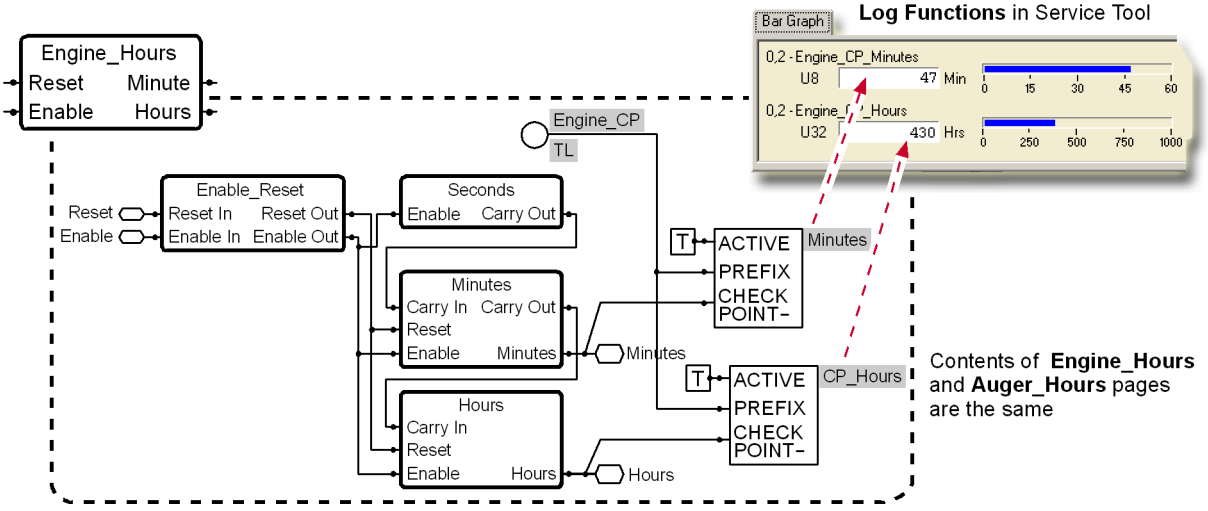
— A3 = False, then the **?** signal is not available

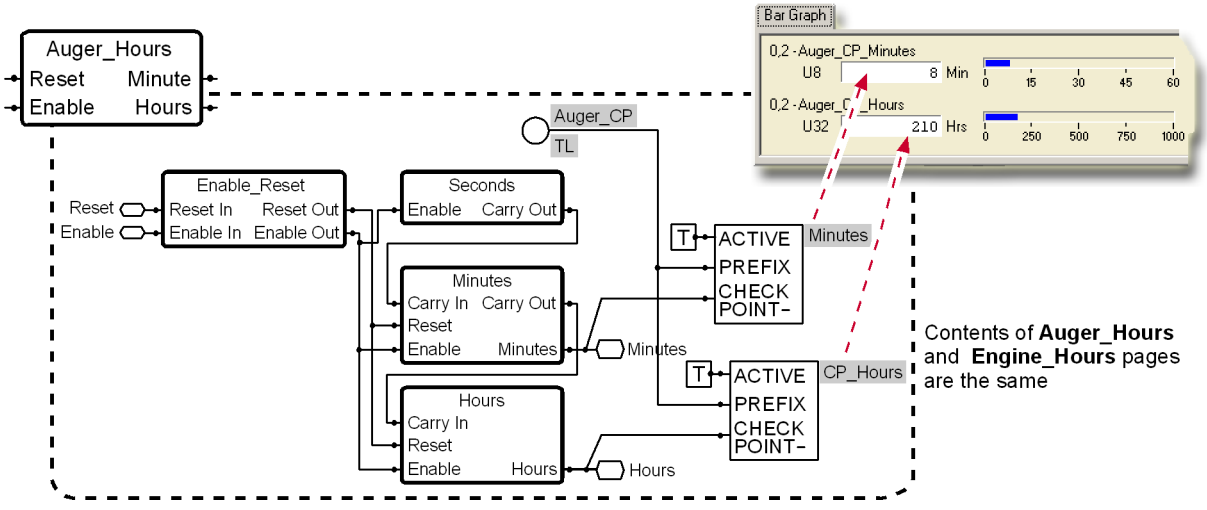
Valid connections

Pin	Data Type	Pin	Data Type
A1	EXTENDED	—	——
A2	TL	—	——
A3	BOOL	—	——

Programming

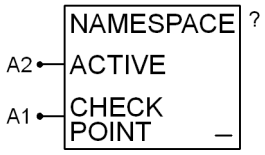
Example—Advanced Checkpoint





Programming

Advanced Checkpoint with Namespace



Use:

- Creates a checkpoint signal that you can check with the PLUS+1 Service Tool program
- Route the signal that you want to check to A1; your **?** entry is the signal name
- Prefixes the page's **Namespace** to the **?** signal name
  - When copying a page, you only have to change the page's **Namespace**
  - You do not have to change **?** names to be able to use the copied page
- Click the underscore (_) with the Query/Change tool to set a 0–9 PLUS+1 Service Tool program access level
- The input to A2 must only be from a **True** or a **False** component; switched True and False inputs cause compiler errors
- The compile process makes this component permanently:
  - Active if a **True** component is connected to its A2 input
  - Inactive if a **False** component is connected to its A2 input
- Cannot be switched between active and inactive states by the Service Tool program or through an application

The **?** names used in checkpoint and non-volatile dynamic memory components must be different. Identical **?** names produce a MULTI DEFINED SYMBOL compile error.

Function:

- **?** = User-defined name for the A1 signal
- If:
  - A2 = True, then the **?** signal is available through the PLUS+1 Service Tool program

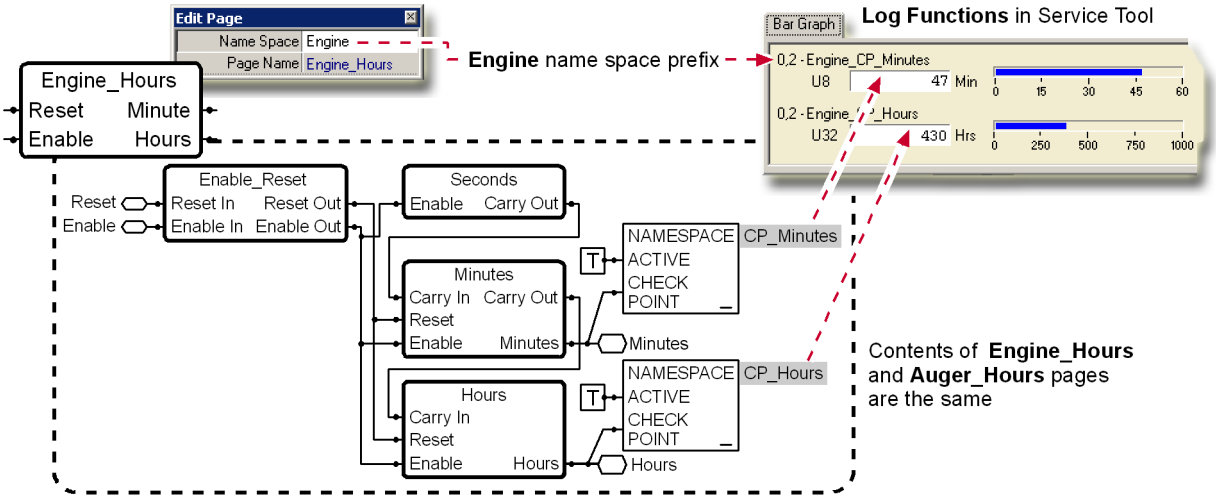
    In the PLUS+1 Service Tool program, the **?** signal name becomes the page's **Namespace** + underscore + **?** signal name
  - A2 = False, then the **?** signal is not available

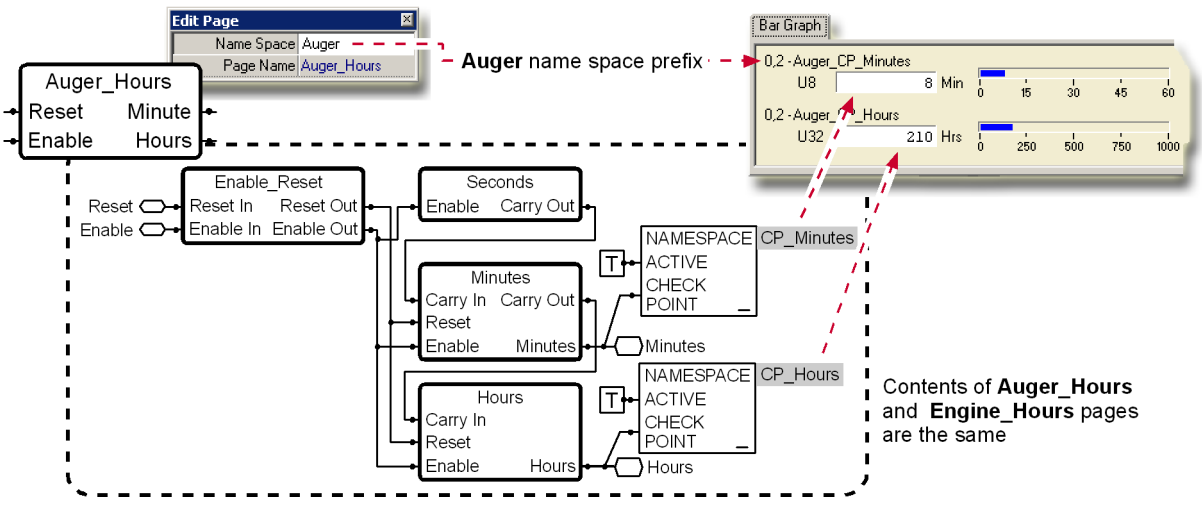
Valid connections

Pin	Data Type	Pin	Data Type
A1	EXTENDED	—	——
A2	BOOL	—	——

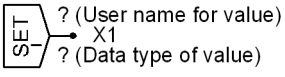
Example—Advanced Checkpoint with Namespace

Programming





Set Value



Use:

- When experimenting with values
- Inputs values directly from the PLUS+1® Service Tool program to the controller without having to use memory components to read and write values
- Turning off the controller returns all values to 0
- Click the underscore (_) with the Query/Change tool to set a 0–9 PLUS+1® Service Tool program access level

Function: ? = User name for the value; use this name to access the value in the PLUS+1® Service Tool program

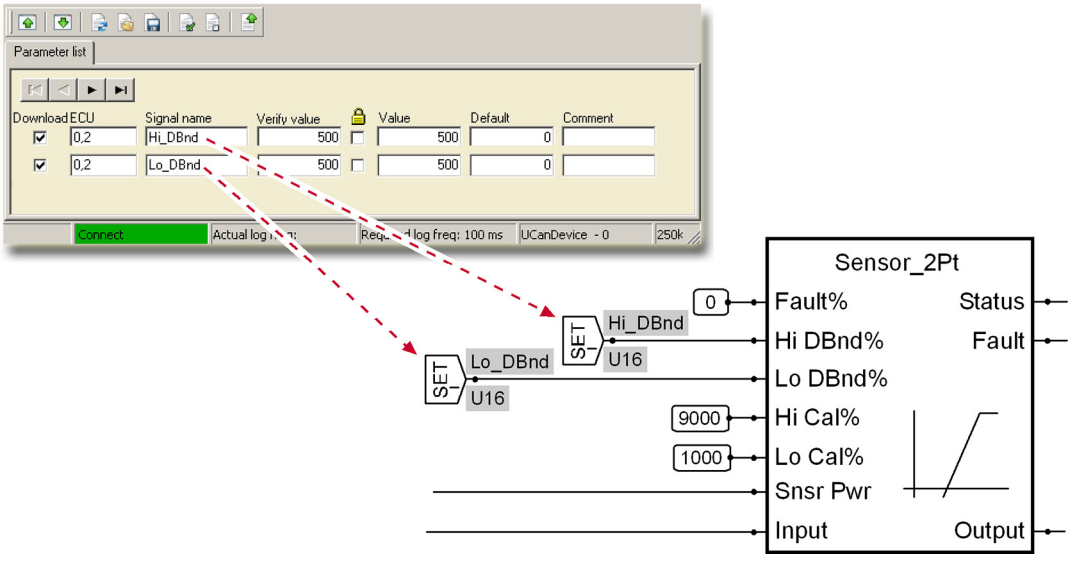
Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	EXTENDED

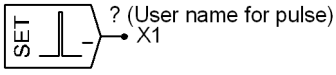
Programming

Example—Set Value

Parameter Functions in Service Tool



Set Pulse



Use:

- Applies a True/False pulse for one program loop when activated through the PLUS+1 Service Tool program
- Simplifies writing calibration values to memory components
- Click the underscore (_) with the Query/Change tool to set a 0–9 PLUS+1 Service Tool program access level

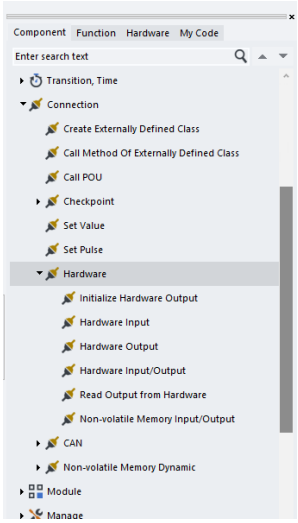
Function: ? = User name for the pulse; link this name with the pulse pushbutton in the PLUS+1 Service Tool program

Valid connections

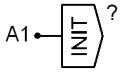
Pin	Data Type	Pin	Data Type
—	——	X1	BOOL

Programming

Hardware Menu



Initialize Hardware Output



Use: Writes a value to the kernel before the first program loop

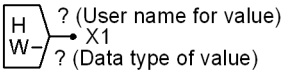
Function:

- **?** = Hardware signal name (must be an exact match)—refer to the Application Interface (API) specification for a list of signals
- Accepts only inputs from constant and non-volatile memory (EE) components
- A changed input value from an EE component takes effect only after the controller repowers

Valid connections

Pin	Data type	Pin	Data type
A1	MAIN, STRING	—	——

Hardware Input Typed



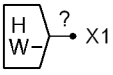
Use: Reads a value from the kernel into the application

Function: **?** = Hardware signal name (must be an exact match)—refer to the Application Interface (API) specification for a list of signals

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	EXTENDED, MAIN

Hardware Input



Use: Reads a value from the kernel into the application

Function:

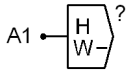
Programming

- Autotyped
- **?** = Hardware signal name (must be an exact match)—refer to the Application Interface (API) specification for a list of signals

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	MAIN, STRING

Hardware Output



Use:

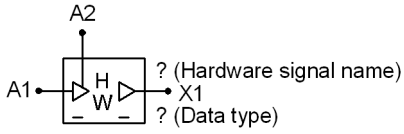
- Writes a value from the application to the kernel at the end of every program loop
- To set an output just one time, before the start of the first program loop, use the **Initialize Hardware Output** component (see *Initialize Hardware Output* on page 320)

Function: **?** = Hardware signal name (must be an exact match)—refer to the Application Interface (API) specification for a list of signals

Valid connections

Pin	Data Type	Pin	Data Type
A1	MAIN, STRING	—	——

Hardware Input/Output Typed



If you are creating a new application using templates developed for use with PLUS+1 GUIDE 2.1(and later), use the new CAN components (see *CAN Menu* on page 325). If you are updating an application that uses this component (typically for CAN communication), see *Using Old CAN and Memory Components* on page 324 .

Use: Switches between reading from or writing to the kernel

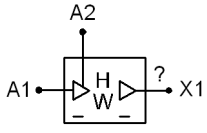
Function:

- **?** = Hardware signal name (must be an exact match)—refer to the Application Interface (API) specification for a list of signals
- X1 = Reading from kernel if A2 = False (X1 not connected to A1)
- X1 = Writing to kernel if A2 = True (A1 = X1)

Valid connections

Pin	Data Type	Pin	Data Type
A1	ALL	X1	MAIN
A2	BOOL	—	——

Hardware Input/Output



Programming

If you are creating a new application using templates developed for use with PLUS+1 GUIDE 2.1(and later), use the new CAN components (see *CAN*). If you are updating an application that uses this component (typically for CAN communication), see *Using Old CAN and Memory Components* .

Use: Switches between reading from or writing to the kernel

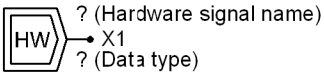
Function:

- Autotyped
- **?** = Hardware signal name (must be an exact match)—refer to the Application Interface (API) specification for a list of signals
- X1 = Reading from kernel if A2 = False (X1 not connected to A1)
- X1 = Writing to kernel if A2 = True (X1 = A1)

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG	X1	BOOL, INT, TIME, FILE, FONT, PORT, COL, IMG
A2	BOOL	—	——

Read Output from Hardware Typed



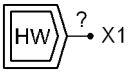
Use: Reads a kernel output signal

Function: ? = Hardware signal name (must be an exact match)—refer to the Application Interface (API) specification for a list of signals

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	MAIN, STRING

Read Output from Hardware



Use:

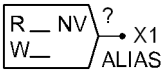
- Reads a kernel output signal
- X1 output is autotyped

Function: ? = Hardware signal name (must be an exact match)—refer to the Application Interface (API) specification for a list of signals

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	MAIN, STRING

Non-Volatile Memory Input



Programming

If you are updating an older application that uses this component, see *Using Old CAN and Memory Components* on page 324.

If you are creating a new application using templates developed for use with PLUS+1® GUIDE 2.1(and later), use the new components, see *Non-Volatile Memory Dynamic Menu* on page 332, for list.

Use: Connection point for an input to non-volatile memory

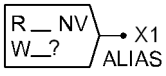
Function:

- **?** = Hardware signal name (must be an exact match)
- Replace the **ALIAS** with a name that you want the PLUS+1® Service Tool program to display. The PLUS +1® Service Tool program uses the hardware name if **ALIAS** is not changed.

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	MAIN

Non-Volatile Memory Input Typed



If you are updating an older application that uses this component, see *Using Old CAN and Memory Components* on page 324.

If you are creating a new application using templates developed for use with PLUS+1 GUIDE 2.1(and later), use the new *Non-Volatile Memory Dynamic Menu* on page 332 components listed.

Use: Connection point for an input to non-volatile memory

Function:

- **?** = Hardware signal name (must be an exact match)

  Hardware signal name must be data typed
- Replace the **ALIAS** with a name that you want the PLUS+1 Service Tool program to display. The PLUS +1 Service Tool program uses the hardware name if **ALIAS** is not changed.

Valid connections

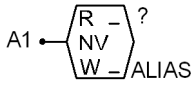
Pin	Data Type	Pin	Data Type
—	——	X1	INT

Example—Non-Volatile Memory Input Typed

X1 outputs the S16 value stored in memory location EE.s16[0].

In the PLUS+1® Service Tool, select the EE_CmpVal signal to view and change the value stored in memory location EE.s16[0].

Non-Volatile Memory Input/Output



If you are updating an older application that uses this component, see *Using Old CAN and Memory Components* on page 324.

If you are creating a new application using templates developed for use with PLUS+1 GUIDE 2.1(and later), use the new *Non-Volatile Memory Dynamic Menu* on page 332 components listed.

Use: Connection point for an input to non-volatile memory

Programming

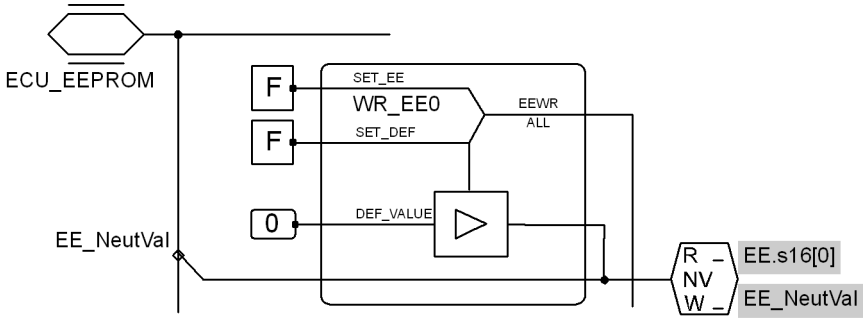
Function:

- A1 = Hardware signal name (must be an exact match)
- **?** = Software location of data
- Replace **ALIAS** with a name that you want the PLUS+1 Service Tool program to display. The PLUS+1 Service Tool program uses the hardware name (EE.s nn[n]) if **ALIAS** is not changed.

Valid connections

Pin	Data Type	Pin	Data Type
A1	MAIN	—	——

Example—Non-Volatile Memory Input/Output



In the Service Tool, select the EE_NeutVal signal to view and change the value stored in memory location EE.s16[0].

Using Old CAN and Memory Components

This table lists old and new CAN and memory (EEprom) components. PLUS+1 GUIDE 2.1 software replaced the **Old** components with the **New** components.

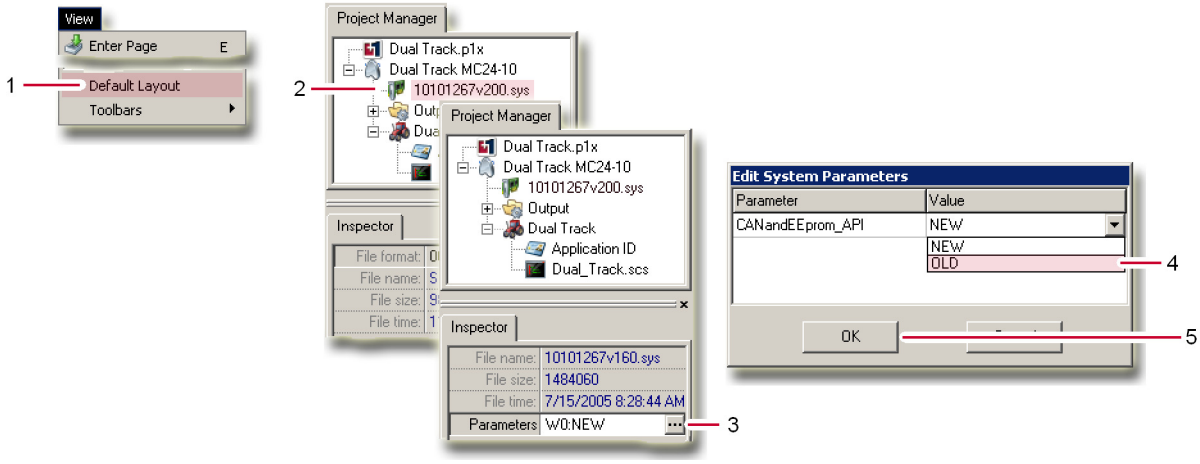
Old and New CAN and Memory components

Old	New
Non-Volatile Memory Input	Transmit CAN
Non-Volatile Memory Input Typed	Receive CAN with Filter
Hardware Input/Output Typed	Receive CAN with ID Mask
Hardware Input/Output Typed	Receive CAN Basic
——	Non-Volatile Memory Dynamic with Default
——	Non-Volatile Memory Dynamic
——	Non-Volatile Memory Input

- Applications created using templates developed for use with PLUS+1 GUIDE 2.1 (and later) software must use the **New** components.
- You must change System file parameters from **NEW** to **OLD** to enable PLUS+1 GUIDE 2.1 (and later) software to compile applications with the **Old** components listed in this table. See *Change from New to Old CAN and Memory Components* .
- PLUS+1 GUIDE 2.1(and later) software with System file parameters set to **OLD** cannot compile applications with the **New** components listed in this table.

Programming

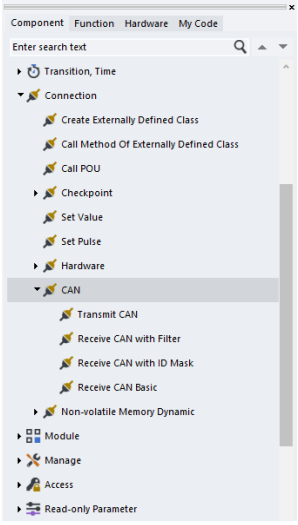
Change from New to Old CAN and Memory Components



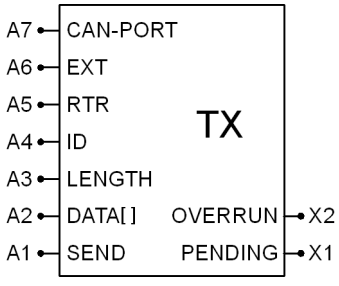
To change the System file parameters from **NEW** to **OLD**:

1. In the **View** menu, click **Default Layout** to display the **Project Manager** and **Inspector** tabs.
2. In the **Project Manager** tab, click the System file.
3. In the **Inspector** tab, click to display the **Edit System Parameters** window.
4. In the **Edit System Parameters** window, click **OLD**.
5. In the **Edit System Parameters** window, click **OK**.

CAN Menu



Transmit CAN



Programming

If you are updating an application that was created before the release of PLUS+1® GUIDE 2.1 software, see *Using Old CAN and Memory Components* on page 324 before using this component. This component, when used in a module, continues to transmit if its A1 (SEND) = False when the module is disabled. To stop this, use logic to keep the module enabled until A1 = False.

Use: Transmits CAN messages.

Function:

- A1 = Message on A2 transmits when True
- A2 = Array of message data field, read from data byte 0
- A3 = DLC (Data length code), number of bytes (0–8) in the data field
  - Output DLC will be the minimum of A3 and the length of the array connected to A2
- A4 = Message ID (identifier field)
- A5—if:
  - A5 = True, RTR (Remote Transmission Request flag set (not supported in J1939)
  - A5 = False, RTR flag not set
- A6—if:
  - A6 = False, standard 11-bit message ID field (CAN 2.0A)
  - A6 = True, extended 29-bit message ID field (CAN 2.0B)
- A7—connect to the CAN port defined by the hardware file—refer to your hardware's Application Interface (API) specification
- X1—if:
  - X1 = True, message pending
  - X1 = False, no message pending
- X2—if:
  - X2 = False, no overrun
  - X2 = True, new message (A1 = True), with pending message not sent (X1 = True); messages not sent are lost

When transmitting an RTR message, the length of the array connected to A2 will be considered even though no data will be transmitted. The DLC of an RTR message will be set to the minimum of A3 and the length of the array connected to A2.

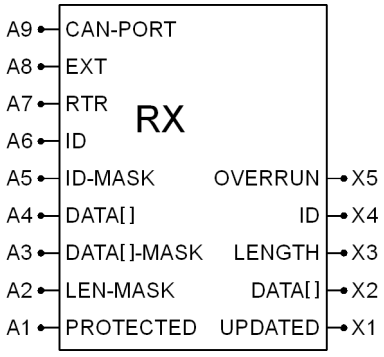
Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	ARRAY[]U8	X2	BOOL
A3	INT	—	——
A4	UINT	—	——
A5	BOOL	—	——
A6	BOOL	—	——
A7	PORT	—	——

The *Example—Receive CAN Basic* on page 330 uses this component.

Programming

Receive CAN with Filter



If you are updating an application that was created before the release of PLUS+1 GUIDE 2.1 software, see *Using Old CAN and Memory Components* on page 324 before using this component.

Use:

- Receive CAN messages
- Apply masking to ID input
- Apply masking to DATA input

Function:

- A1—if:
  - A1 = True, saves the first message received in the program loop; following messages do not overwrite the first message
  - A1 = False, saves the last message received in the program loop; the last message overwrites any preceding message
- A2 = Number of bytes in **DATA[ ]-MASK** to use
- A3 = Mask applied to A4 data
- A4 = Data
- A5 = Mask applied to message ID (identifier field)
- A6 = Message ID (identifier field)
- A7—if:
  - A7 = True, receives only transmission request (RTR) messages
  - A7 = False, does not receive RTR messages
- A8—if:
  - A8 = False, standard 11-bit message ID field (CAN 2.0A)
  - A8 = True, extended 29-bit message ID field (CAN 2.0B)
- A9—connect to the port signal on the CAN bus as defined by the hardware file:
- X1—if:
  - X1 = True, new message received since last program loop
  - X1 = False, no new messages received since last program loop
- X2 = Array of message data field, read from data byte 0 (Note: The number of valid indexes in this array is given by the value of X3)
- X3 = Data length code, number of bytes (0–8) in the data field
- X4 = Message ID
- X5—if:

Programming

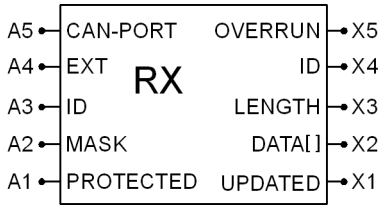
- X5 = True, more than one message received during program loop
- X5 = False, no overrun

- CAN RX components placed in the same module receive their CAN messages in the normal left-to-right, top-to-bottom order
- CAN RX components placed in different modules receive their CAN messages in a top-to-bottom order that follows the order in which the **Project Manager** tab lists the modules

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	UINT	X2	ARRAY[8] U8
A3	ARRAY[ ] U8	X3	U8
A4	ARRAY[ ] U8	X4	U32
A5	UINT	X5	BOOL
A6	UINT	—	——
A7	BOOL	—	——
A8	BOOL	—	——
A9	PORT	—	——

Receive CAN with ID Mask



If you are updating an application that was created before the release of PLUS+1 GUIDE 2.1 software, see *Using Old CAN and Memory Components* on page 324 before using this component.

Use:

- Receive CAN messages
- Apply masking to message ID input

Function:

- A1—if:
  - A1 = True, saves the first message received in the program loop; following messages do not overwrite the first message
  - A1 = False, saves the last message received in the program loop; the last message overwrites any preceding message
- A2 = Defines mask applied to message ID (identifier field); if a mask bit is 0, then it is a “do not care”
- A3 = Message ID (identifier field)
- A4—if:
  - A4 = False, standard 11-bit message ID field (CAN 2.0A)
  - A4 = True, extended 29-bit message ID field (CAN 2.0B)
- A5—connect to the port signal on the CAN bus as defined by the hardware file—refer to your hardware’s Application Interface (API) specification
- X1—if:
  - X1 = True, new message received since last program loop
  - X1 = False, no new messages received since last program loop

Programming

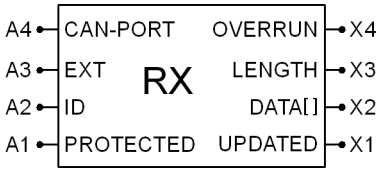
- X2 = Array of message data field, read from data byte 0 (Note: The number of valid indexes in this array is given by the value of X3)
- X3 = Data length code, the number of bytes (0–8) in the data field
- X4 = Actual message ID
- X5—if:
  - X5 = True, more than one message received during program loop
  - X5 = False, no overrun
- CAN RX components placed in the same module receive their CAN messages in the normal left-to-right, top-to-bottom order
- CAN RX components placed in different modules receive their CAN messages in a top-to-bottom order that follows the order in which the **Project Manager** tab lists the modules

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	UINT	X2	ARRAY[8] U8
A3	UINT	X3	U8
A4	BOOL	X4	U32
A5	PORT	X5	BOOL

Programming

Receive CAN Basic



If you are updating an application that was created before the release of PLUS+1 GUIDE 2.1 software, see *Using Old CAN and Memory Components* on page 324 before using this component.

Use: Receives CAN messages.

Function:

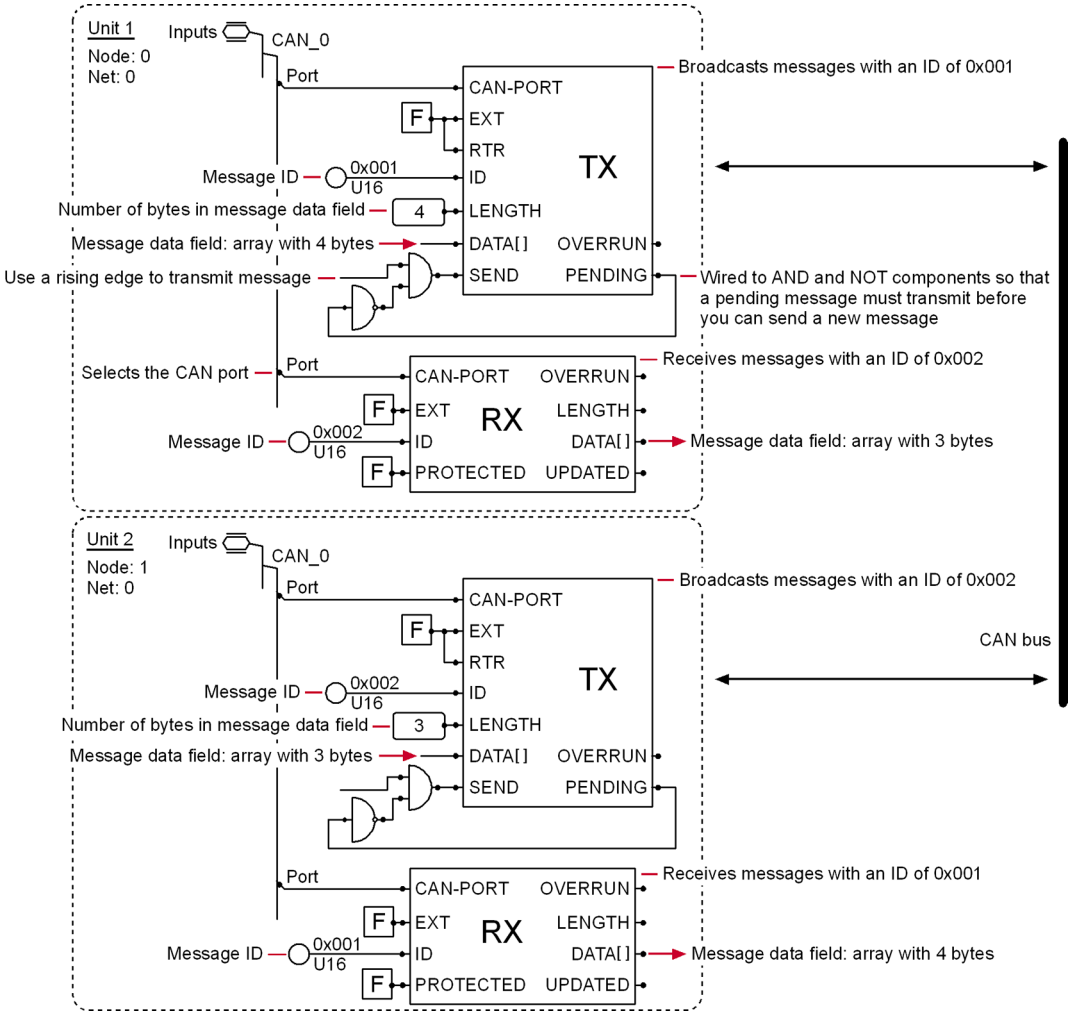
- A1—if:
  - A1 = True, saves the first message received in the program loop; following messages do not overwrite the first message
  - A1 = False, saves the last message received in the program loop; the last message overwrites any preceding message
- A2 = Message ID (identifier field)
- A3—if:
  - A3 = False, standard 11-bit message ID field (CAN 2.0A)
  - A3 = True, extended 29-bit message ID field (CAN 2.0B)
- A4—connect to the port signal on the CAN bus as defined by the hardware file:
  - CAN_0 bus for MC024 Microcontroller
  - CAN_0 bus or CAN_1 bus for MC050 Microcontroller
- X1—if:
  - X1 = True, new message received since last program loop
  - X1 = False, no new messages received since last program loop
- X2 = Array of message data field, read from data byte 0 (Note: The number of valid indexes in this array is given by the value of X3)
- X3 = Data length code, number of bytes (0–8) in the data field
- X4—if:
  - X4 = True, more than one message received during program loop
  - X 4 = False, no overrun
- CAN RX components placed in the same module receive their CAN messages in the normal left-to-right, top-to-bottom order
- CAN RX components placed in different modules receive their CAN messages in a top-to-bottom order that follows the order in which the **Project Manager** tab lists the modules

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	BOOL
A2	UINT	X2	ARRAY[8]U8
A3	BOOL	X3	U8
A4	PORT	X4	BOOL

Example—Receive CAN Basic

Programming



In the Unit 1:

- the **Transmit CAN** component (TX) broadcasts messages with an ID of 0x001
- the **Receive CAN Basic** component (RX) receives messages with an ID of 0x002

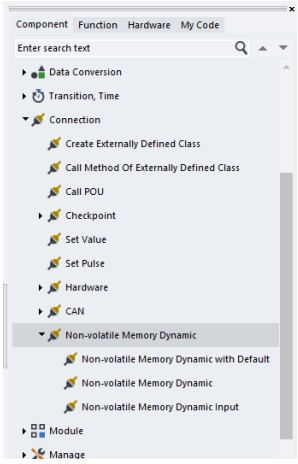
In the Unit 2:

- the **Transmit CAN** component (TX) broadcasts messages with an ID of 0x002
- the **Receive CAN Basic** component (RX) receives messages with an ID of 0x001

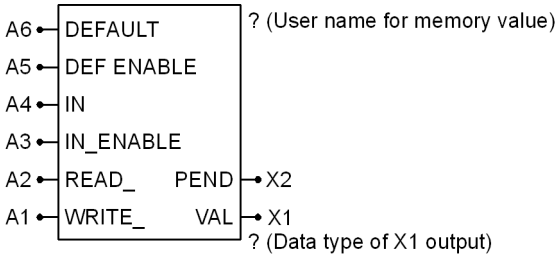
For information about using **Set Array Element** components to construct a message data field that contains dynamic run-time values, see *Set Array Element* on page 267 and **Get Array Element** components to output individual elements within a message data field, see *Get Array Element* on page 266.

Programming

Non-Volatile Memory Dynamic Menu



Non-Volatile Memory Dynamic with Default



Warning

All non-volatile memory has a limit to the total number of times that you can erase from and write to a memory sector. Non-volatile memory can become corrupted when you exceed this erase/write limit.

This corruption can change the values that your application reads from non-volatile memory. Changed values in your application can cause sudden, unexpected machine movements, equipment damage, and personal injury.

To reduce the risk of non-volatile memory corruption, you should:

- Refer to *PLUS+1 Controller Family Technical Information* BC152886484710 to find the lifetime erase/write rating of the non-volatile memory that is used in your controller.
- Develop your application so that no value written to non-volatile memory will exceed this erase/write rating during the lifetime of your application.
- Use a **Positive Transition** component (as shown in the following example) with this memory component to prevent accidental continuous erase/write cycles.

If you are updating an application that was created before the release of PLUS+1 GUIDE 2.1 software, first see *Using Old CAN and Memory Components*.

Use:

- Read and write values to non-volatile (NV) memory
- Write default values to NV memory
- On hardware power-up, the value in NV memory copies to X1

Function:

- **?** = Your name for the NV memory value

Programming

  - When used in a page that has a **Namespace** value, the name of the NV memory value will be prefixed with the **Namespace** value

- If A1 is True and X2 is False, then the X1 value is written to NV memory
- If A2 is True, then the value in NV memory is copied to X1
- If A3 is True and both A2 and A5 are False, then the A4 value is copied to X1, independent of X2 status
  - A3 has no effect if either A2 or A5 are True
- A4 is the New value to use for X1 if A3 is True and both A2 and A5 are False
- If A5 is True and A2 is False, then the A6 value is copied to X1, independent of X2 status
  - A5 has no effect if A2 is True
- A6 is the Default value to use for X1 if A5 is True and A2 is False
- X1 is the Value read from NV memory/value to be written to NV memory
- X2 = True indicates that writing of X1 value to NV memory is pending completion

Service Tool interaction:

- If the read access level permits, the Service Tool can read from the NV memory using the Name set on this component (including **Namespace** if any).
  - The Service Tool will read the value from NV memory, **not X1**.
- If the write access level permits, the Service Tool can write to the NV memory using the Name set on this component (including **Namespace** if any).
  - **Warning**

    The value will not only be written to the NV memory, **X1 will also be modified in this case to match the new content in NV memory.** This modification of X1 happens in the very beginning of the loop, before the component is executed.
- In conclusion: reading and writing this NV memory parameter from the Service Tool is not handled in a symmetrical fashion, and you must be aware of this difference when designing the application.

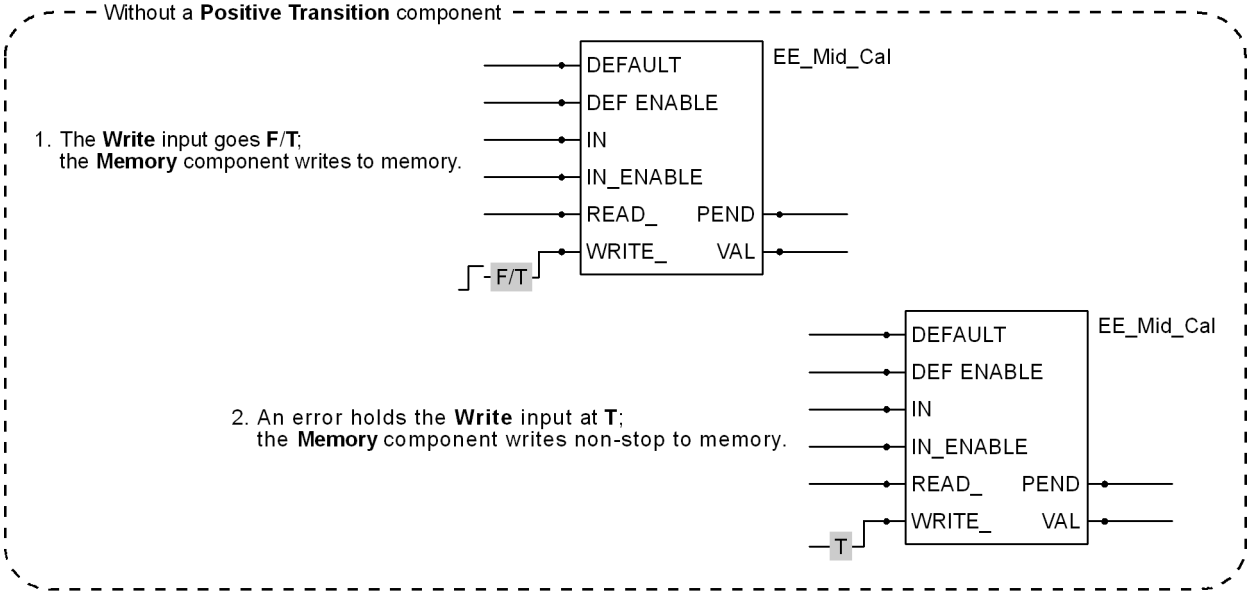
Valid connections

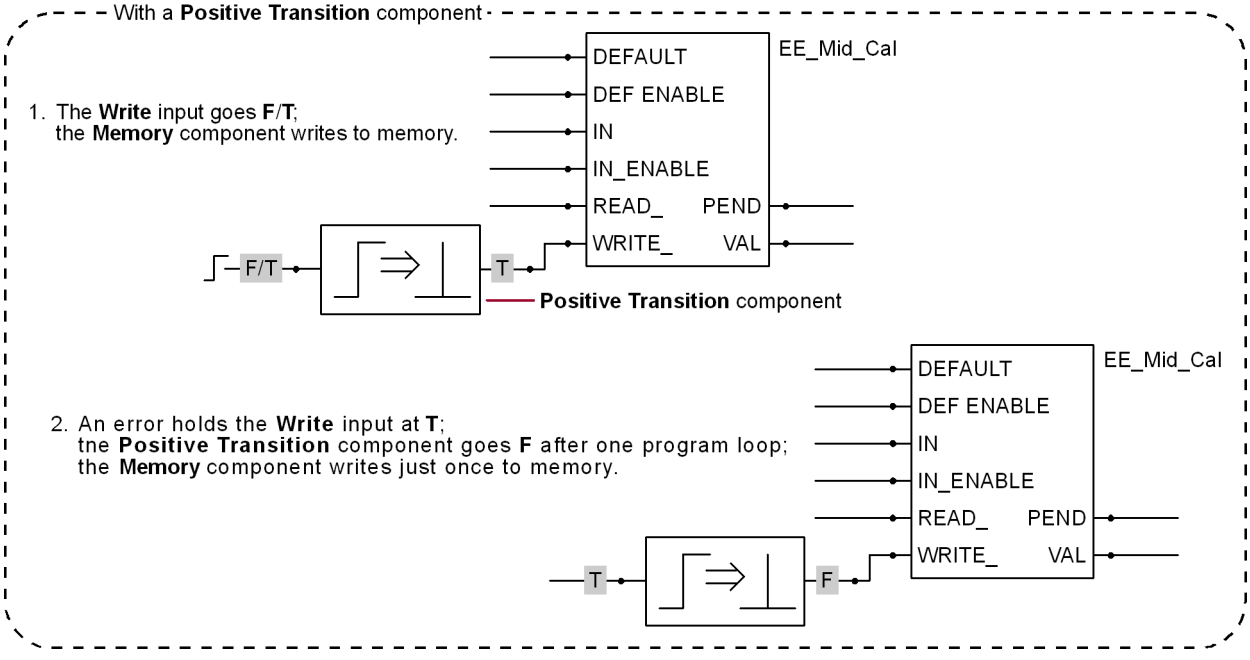
Pin	Data Type	Pin	Data Type
A1	BOOL	X1	EXTENDED
A2	BOOL	X2	BOOL
A3	BOOL	—	——
A4	INT, BOOL	—	——
A5	BOOL	—	——
A6	INT, BOOL	—	——

Programming

Example 1–How to Prevent Accidental Continuous Erase/Write Cycles

Use the **Positive Transition** component with this memory component to prevent accidental continuous erase/write cycles that shorten the life of non-volatile memory.

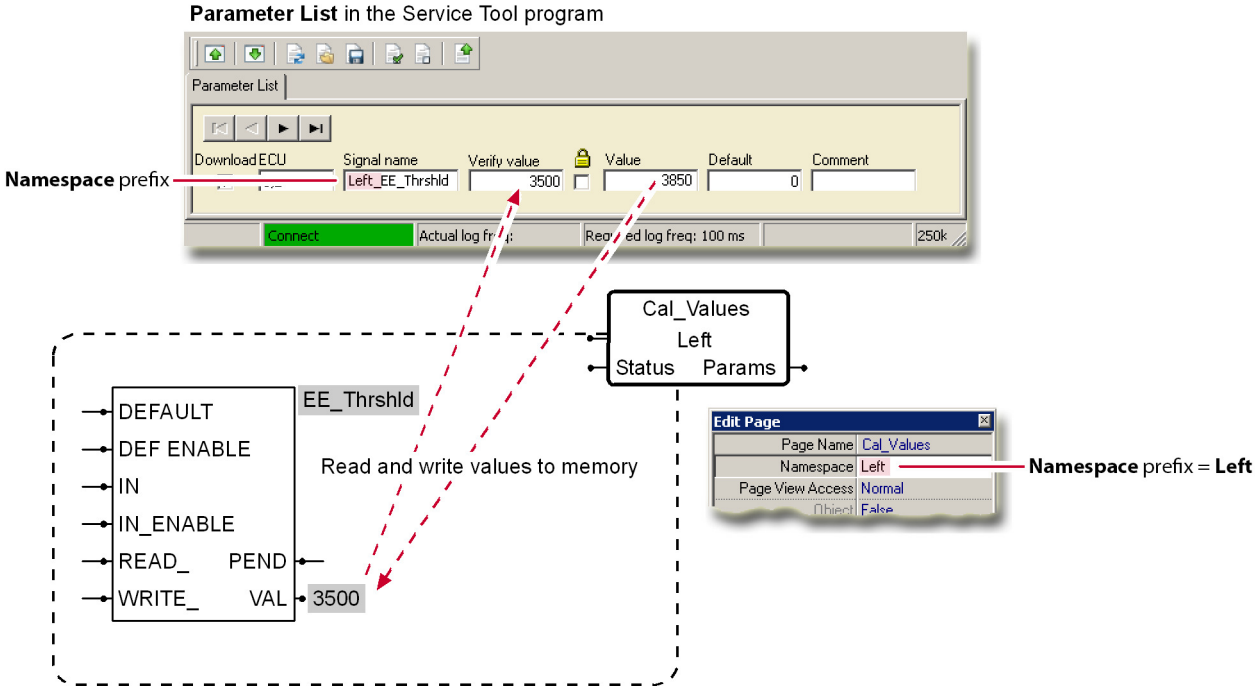


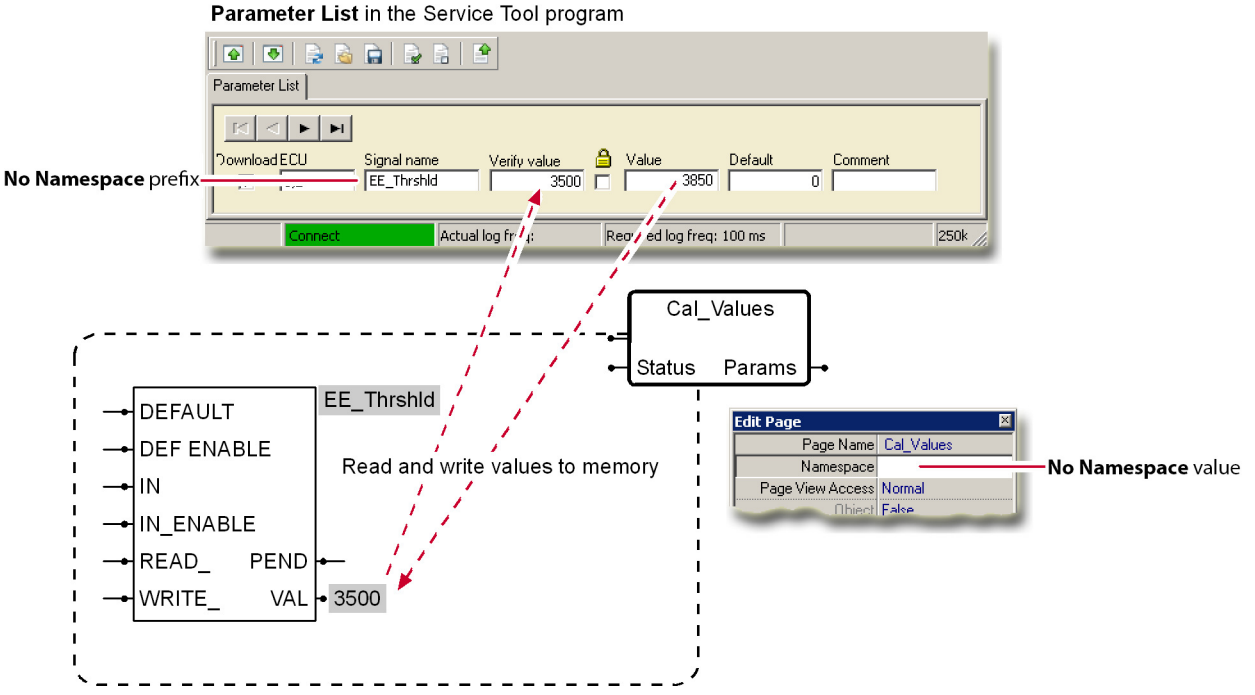


Programming

Example 2—Non-Volatile Memory Dynamic with Default

When this component is used in a page with a **Namespace** value, the PLUS+1® Service Tool program prefixes the name of the memory value with the page's **Namespace** value.

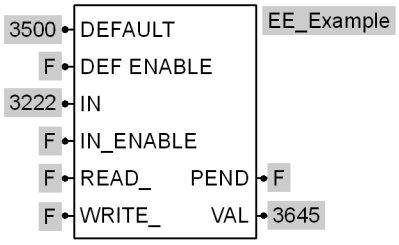




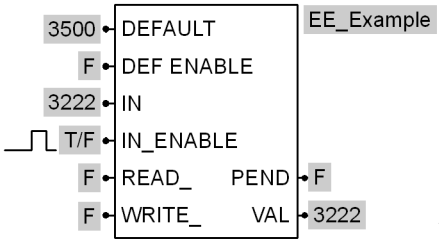
Programming

Example 3—Non-Volatile Memory Dynamic with Default

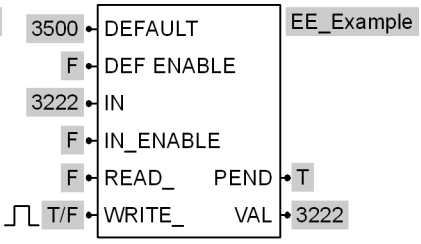
1 Power-up—Value in memory copies to **VAL**



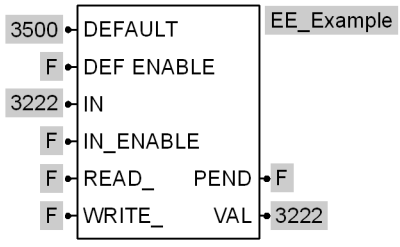
2 **IN** copies to **VAL**



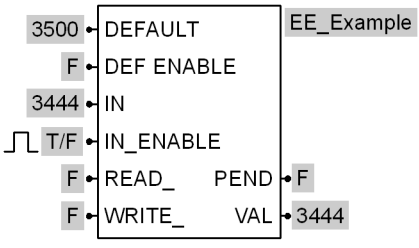
3 **VAL** writes to memory; **PEND** goes **T**



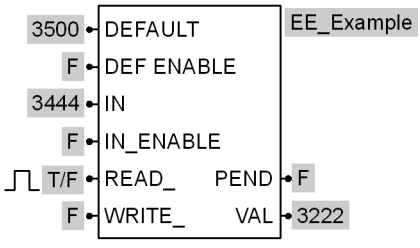
4 Write ends; **PEND** goes **F**



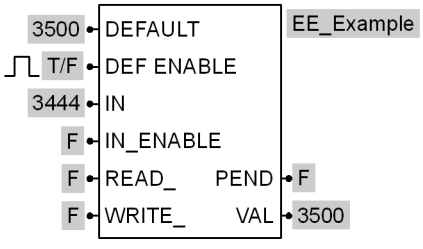
1 **IN** copies to **VAL**



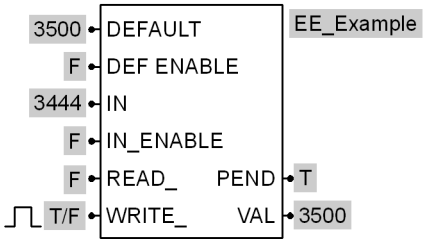
2 Value in memory copies back to **VAL**



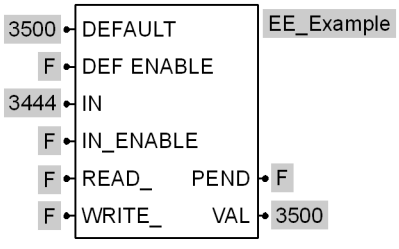
1 **DEFAULT** copies to **VAL**



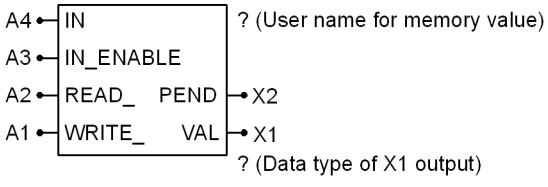
2 **VAL** writes to memory; **PEND** goes **T**



3 Write ends; **PEND** goes **F**



Non-Volatile Memory Dynamic



Programming

Warning

All non-volatile memory has a limit to the total number of times that you can erase from and write to a memory sector. Non-volatile memory can become corrupted when you exceed this erase/write limit.

This corruption can change the values that your application reads from non-volatile memory. Changed values in your application can cause sudden, unexpected machine movements, equipment damage, and personal injury.

To reduce the risk of non-volatile memory corruption, you should:

- Refer to *PLUS+1 Controller Family Technical Information* BC152886484710 to find the lifetime erase/write rating of the non-volatile memory that is used in your controller.
- Develop your application so that no value written to non-volatile memory will exceed this erase/write rating during the lifetime of your application.
- Use a **Positive Transition** component (as shown in the following example) with this memory component to prevent accidental continuous erase/write cycles.

If you are updating an application that was created before the release of PLUS+1 GUIDE 2.1 software, see *Using Old CAN and Memory Components* on page 324 before using this component.

Use:

- Read and write values to non-volatile (NV) memory
- On hardware power-up, the value in NV memory copies to X1

Function:

- **?** = Your name for the NV memory value
  - When used in a page that has a **Namespace** value, the name of the NV memory value will be prefixed with the **Namespace** value
- If A1 is True and X2 is False, then the X1 value is written to NV memory
- If A2 is True, then the value in NV memory is copied to X1
- If A3 is True and A2 is False, then the A4 value is copied to X1, independent of X2 status
  - A3 has no effect if A2 is True
- A4 is the New value for X1 to use if A3 is True and A2 is False
- X1 is the Value read from NV memory/value to be written to NV memory
- X2 = True indicates that writing of X1 value to NV memory is pending completion

Service Tool interaction:

- If the read access level permits, the Service Tool can read from the NV memory using the Name set on this component (including **Namespace** if any).

The Service Tool will read the value from NV memory, **not X1**.

- If the write access level permits, the Service Tool can write to the NV memory using the Name set on this component (including **Namespace** if any).

Warning

The value will not only be written to the NV memory, **X1 will also be modified in this case to match the new content in NV memory.** This modification of X1 happens in the very beginning of the loop, before the component is executed.

Programming

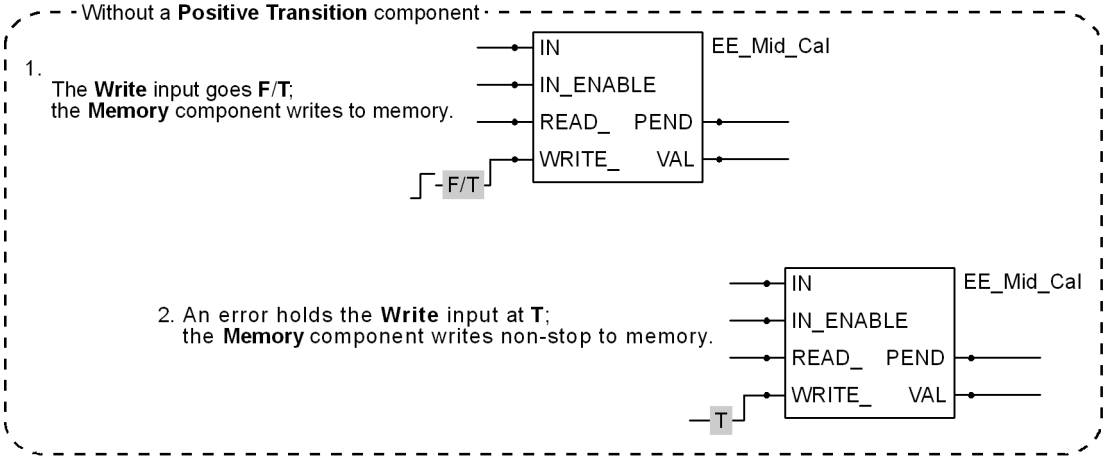
- In conclusion: reading and writing this NV memory parameter from the Service Tool is not handled in a symmetrical fashion, and you must be aware of this difference when designing the application.

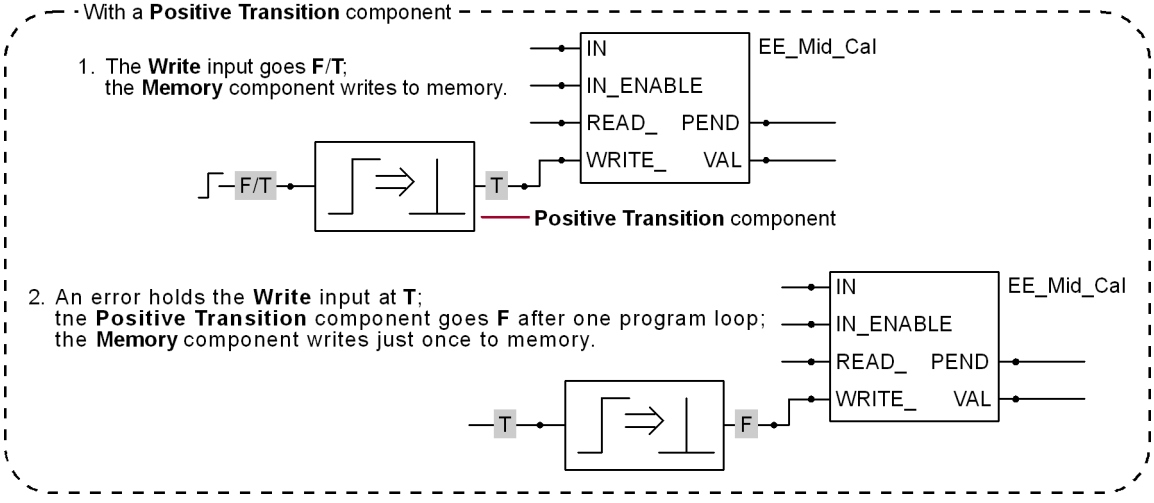
Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	EXTENDED
A2	BOOL	X2	BOOL
A3	BOOL	—	——
A4	INT, BOOL	—	——

Example 1–How to Prevent Accidental Continuous Erase/Write Cycles

Use the **Positive Transition** component with this memory component to prevent accidental continuous erase/write cycles that shorten the life of non-volatile memory.

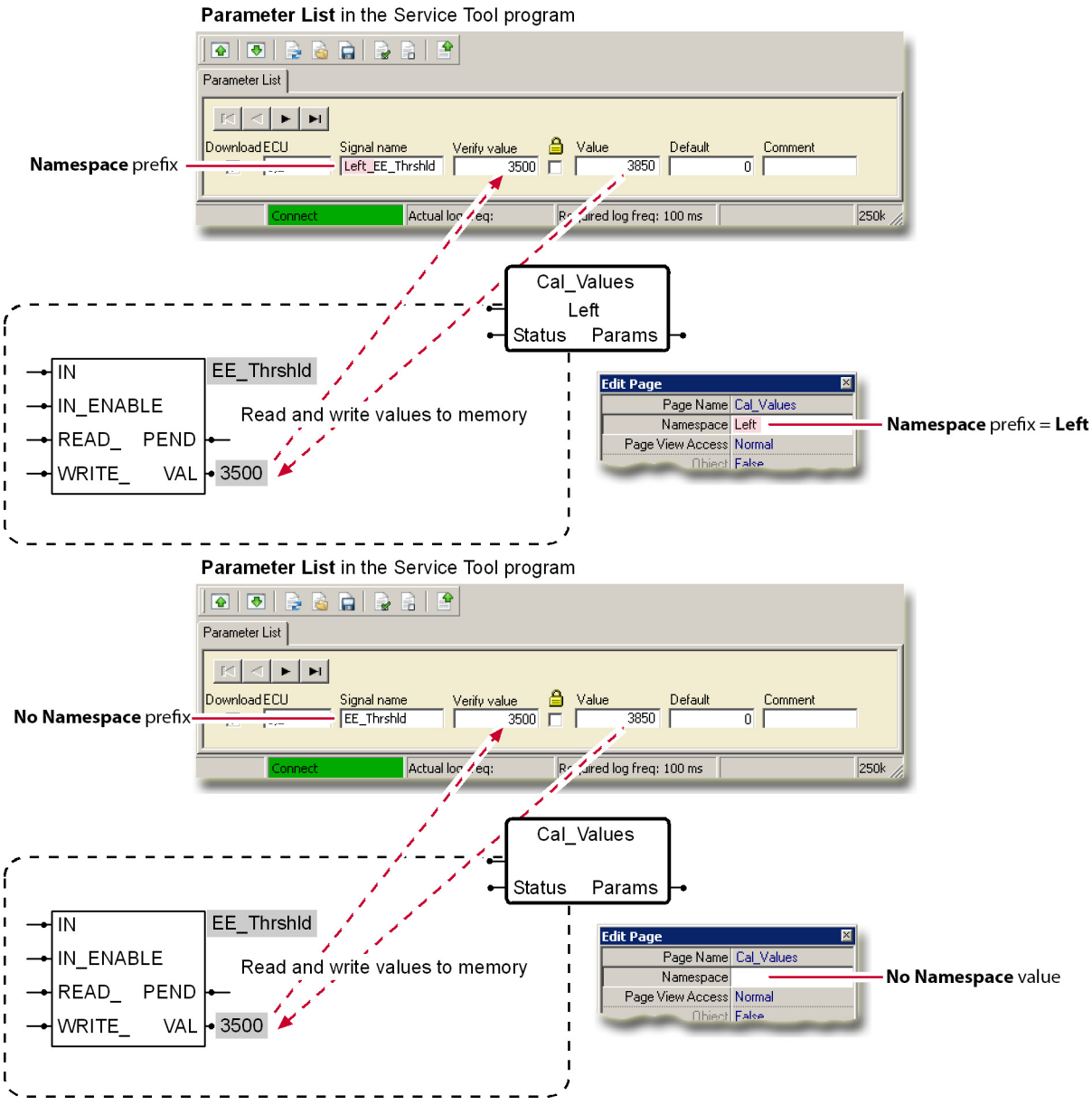




Example 2—Non-Volatile Memory Dynamic

When this component is used in a page with a **Namespace** value, the PLUS+1® Service Tool program prefixes the name of the memory value with the page's **Namespace** value.

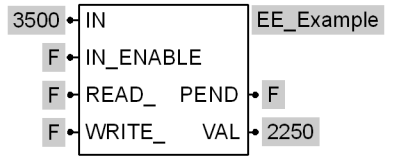
Programming



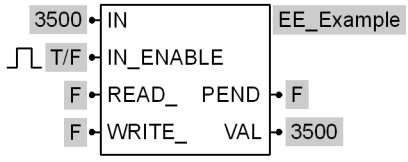
Example 3—Non-Volatile Memory Dynamic

Programming

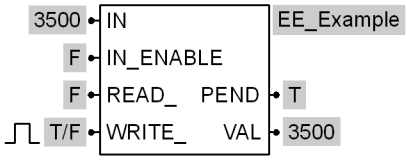
1 Power-up—Value in memory copies to **VAL**



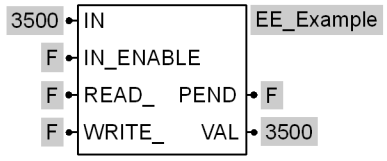
2 **IN** copies to **VAL**



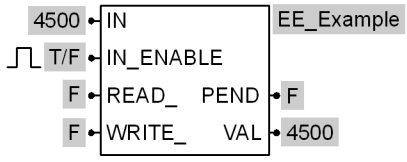
3 **VAL** writes to memory; **PEND** goes **T**



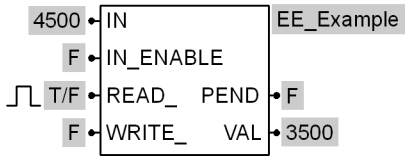
4 Write ends, **PEND** goes **F**



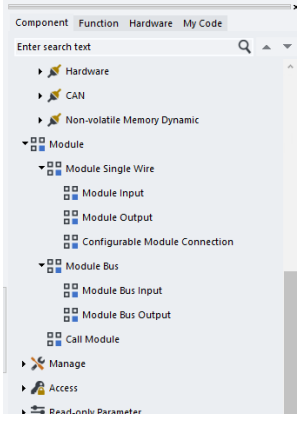
1 **IN** copies to **VAL**



2 Value in memory copies to **VAL**

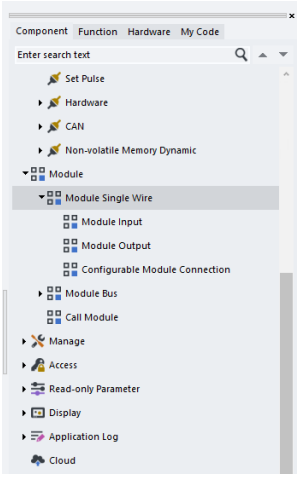


Module Menu

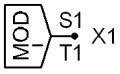


Programming

Module Single Wire Menu



Module Input Typed



Use:

- Input point for a module signal
- Click the underscore (_) with the Query/Change tool to set a 0–9 PLUS+1 Service Tool program access level

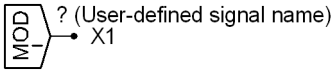
Function:

- **?** = User-defined signal name
- X1 = Input signal

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	EXTENDED, MAIN

Module Input



Use:

- Input point for a module signal
- Click the underscore (_) with the Query/Change tool to set a 0–9 PLUS+1 Service Tool program access level

Function:

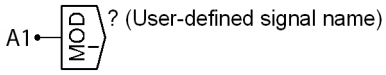
- **?** = User-defined signal name
- X1 = Input signal

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	EXTENDED, MAIN

Programming

Module Output



Use: Output for a module signal

Function:

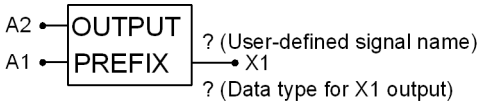
- **?** = User-defined signal name
- A1 = Input signal

Valid connections

Pin	Data Type	Pin	Data Type
A1	EXTENDED, MAIN	——	—

See *Example 1—Call Module* on page 345 of how to use this component.

Configurable Module Connection



Use:

- Makes a connection between two modules:
  - Outputs a signal from a module
  - Inputs the same signal to another module. Use the **Export Block** and **Import Block** commands to export and import connections.

Function:

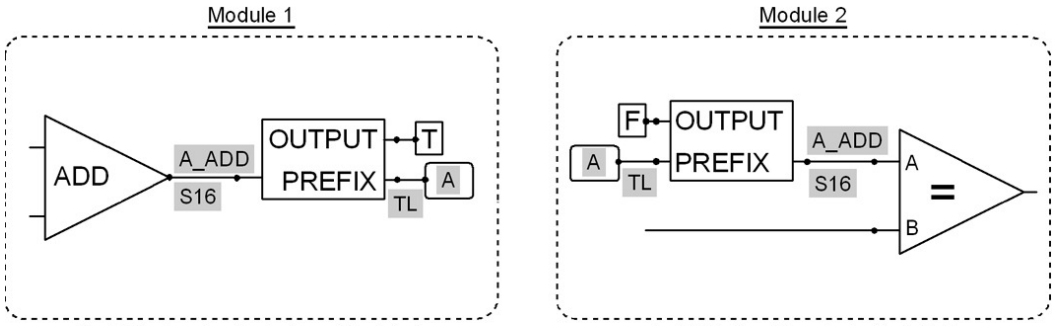
- **?** = User-defined signal name
- If:
  - A2 = True, then X1 = Output signal to another module
  - A2 = False, then X1 = Input signal from another module
  - X1 signal name is the value of A1 + underscore (_) + name of X1; the signal name becomes global

Valid connections

Pin	Data Type	Pin	Data Type
A1	TL	X1	MAIN
A2	BOOL	—	——

Programming

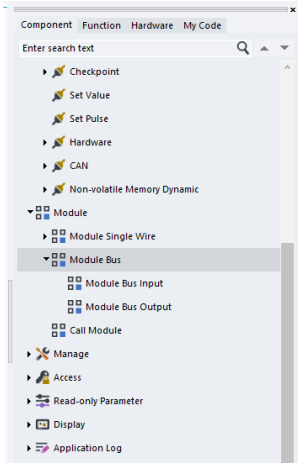
Example—Configurable Module Connection



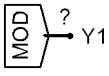
A_ADD signal outputs from Module 1 A_ADD signal inputs to Module 2

When placing or moving components, press F8 to reflect them180°

Module Bus Menu



Module Bus Input



Use: Input for a module bus

Function:

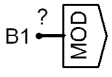
- **?** = User-defined bus name
- Y1 = Bus output

Valid connections

Pin	Data Type	Pin	Data Type
—	——	Y1	EXTENDED, MAIN

The PLUS+1 GUIDE application does not support sub-buses inside module buses. See *Example 2—Call Module* on page 346 for an example that uses this component.

Module Bus Output



Use: Output for a module bus

Function:

Programming

- **?** = User-defined bus name
- B1 = Bus input

Valid connections

Pin	Data Type	Pin	Data Type
B1	EXTENDED, MAIN	—	——

The PLUS+1® GUIDE application does not support sub-buses inside module buses. See *Example 2—Call Module* on page 346 for an example that uses this component.

Call Module

Use: Executes a module, without parameters in the call, contained in another file

Function:

- Replace **NAME ?** with a user-defined module name. The name that you enter must match the **Module name** for the **Module.scs** file.

  (The **Module name** is a property of the **Module.scs** file; the **Inspector** tab displays this **Module name** when you click the **Module.scs** file in the **Project Manager** tab.)
- Replace **COMMENT** with a user comment string.
- If:
  - A1 = True components in the module identified by **NAME ?** execute
  - A1 = False no components in the module identified by **NAME ?** execute
- Do not call the same function more than once in the main loop.
- If the module is not called, module output values remain set.
- Whatever output values apply when A1 goes from True/False, also apply when A1 goes from False/True.
- Each call overwrites existing values in the memory.

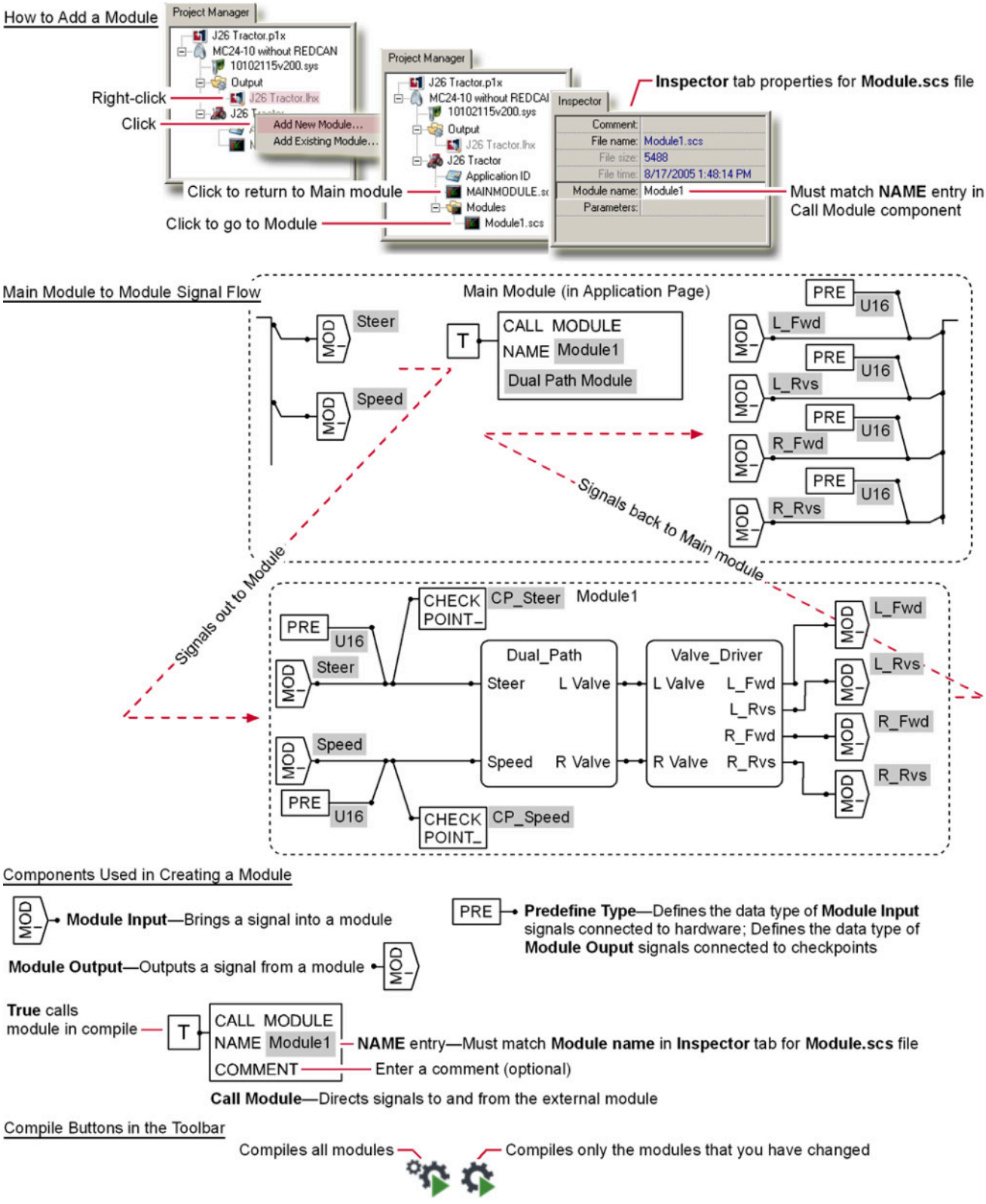
When a module call ends (A1= False), the values that the module outputs when the call ends continue to be output. If your module controls outputs, create logic to turn these outputs off before the module call ends. If your module has CAN transmit components, create logic to turn transmission off before the module call ends.

See the examples *Example 1—Call Module* on page 345 and *Example 2—Call Module* on page 346 of how to use this component.

Programming

Example 1—Call Module

How to use the Module Input Typed, Module Output, and Call Module components to create a module

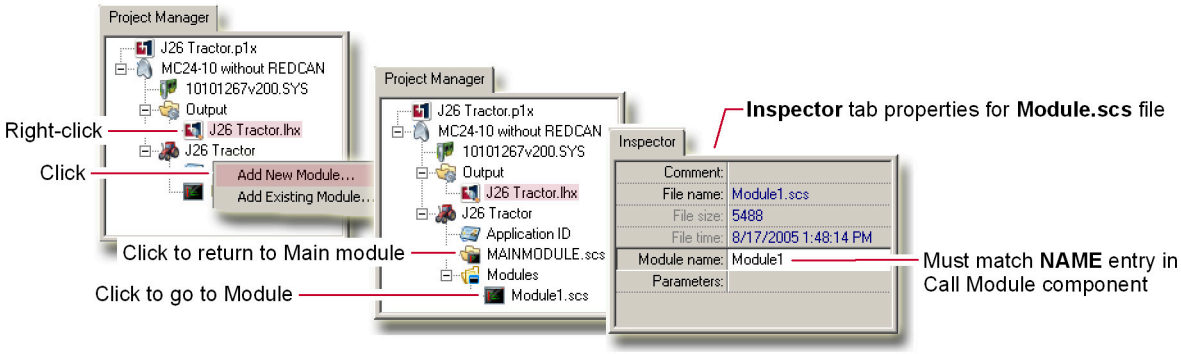


Programming

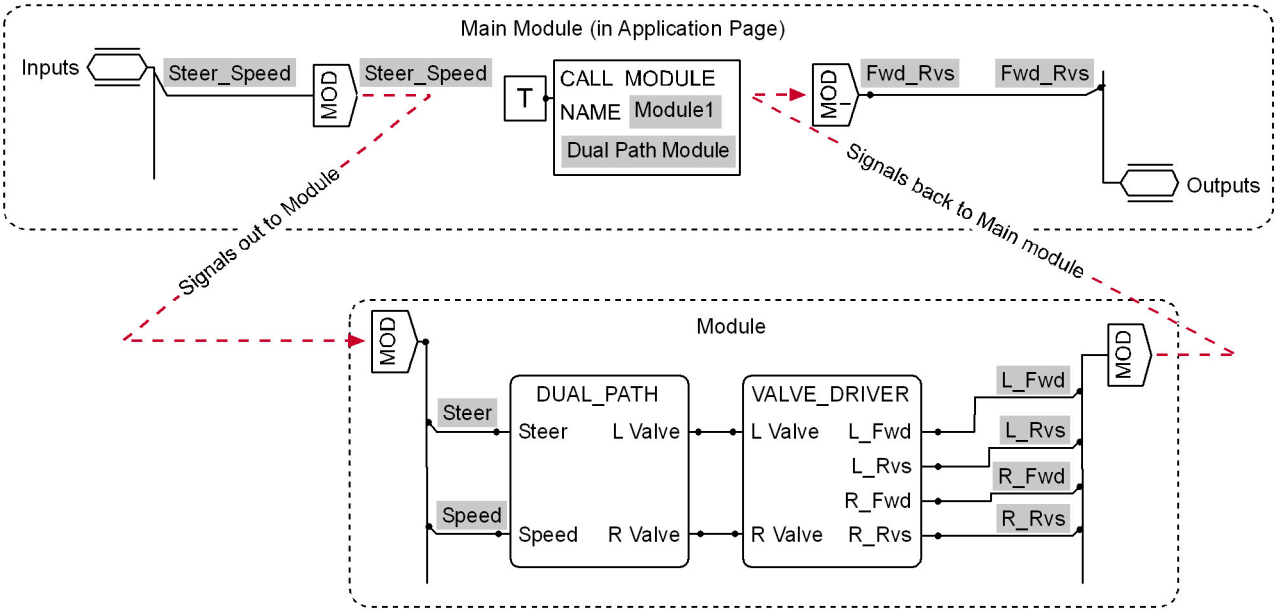
Example 2—Call Module

*How to use the **Module Bus Input**, **Module Bus Output**, and **Call Module** components to create a module.*

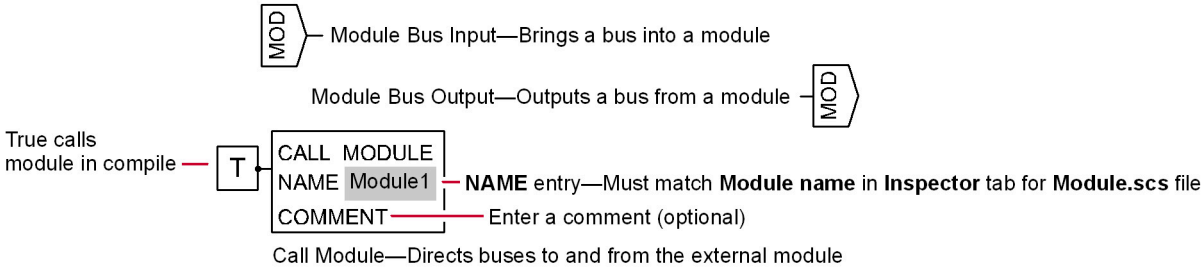
How to Add a Module



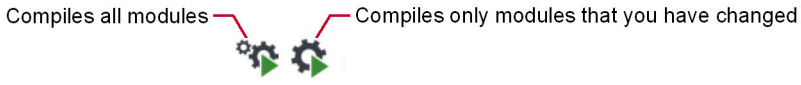
Main Module to Module1 Signal Flow



Components Used in Creating a Module

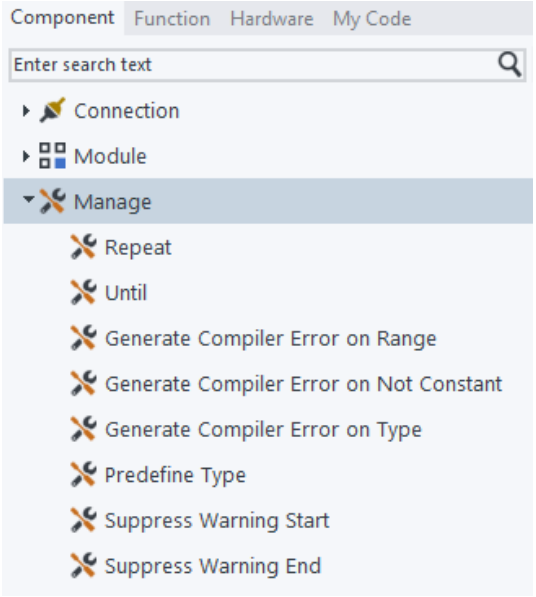


Compile Buttons in Toolbar



Programming

Manage Menu



Suppress Warning Start

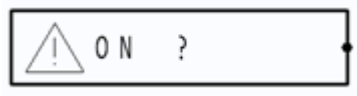
Use:

- Suppresses a specified warning within a code area from being reported by the **Static Analyzer**.

Function:

- The left dot on the component marks the start of the code area.
- Pair with the **Suppress Warning End** component (see *Suppress Warning End* on page 347).

Suppress Warning End



Use:

- Enables a specified warning to be reported by the **Static Analyzer** again.

Function:

- The right dot on the component marks the end of the code area.
- Pair with the **Suppress Warning Start** component (see *Suppress Warning Start* on page 347).

Repeat

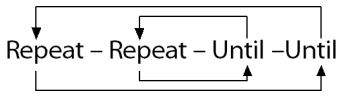
Use:

Programming

- Repeated execution of components within a single application loop
- Marks the start of a repeat-until loop; the **Until** component marks the end of the loop
- Helpful when reading and writing arrays

Function:

- The left dot on the component marks the start of the repeat-until loop
- Pair with the **Until** component (see *Until* on page 348) The **Repeat** and **Until** components must be on the same page; the PLUS+1 GUIDE software does not allow a repeat-until loop across multiple pages.
- X1 = True during the first iteration of the current repeat-until loop in each application loop, then False; if the repeat-until loop finishes after just one iteration, this value remains True outside the repeat-until loop
- Nested repeat-until loops are allowed



- Some items are not allowed inside repeat-until loops; see *Do Not Use Inside Repeat-Until Loops* on page 349

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	BOOL

Until

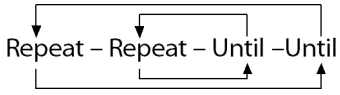
Use:

- Repeated execution of components within a single application loop
- Marks the end of a repeat-until loop; the **Repeat** component marks the start of the loop

Function:

- Pair with the **Repeat** component (see *Repeat* on page 347.) The **Repeat** and **Until** components must be on the same page; the PLUS+1 GUIDE software does not allow a repeat-until loop across multiple pages
- A1 = True—end the repeat-until loop
- X1 = True when a timeout condition is aborting the repeat-until loop
  - The controller checks for a timeout condition each time the **Until** component executes
  - A timeout condition occurs if the application loop time becomes greater than the controller's **OS.ExecTimeOut**
  - A timeout condition aborts the repeat-until loop; any following repeat-until loops execute once and then abort

    (A timeout condition can occur if the controller reaches the **OS.ExecTimeOut** on the last wanted iteration of a repeat-until loop; even though the repeat-until loop completely executes, the **Until** component's X1 output goes True to indicate a timeout condition)
- Nested repeat-until loops are allowed



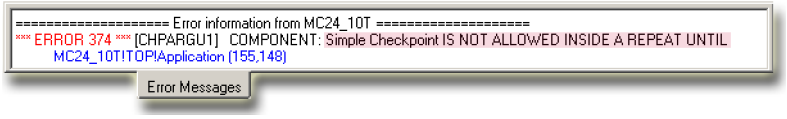
- Some items are not allowed inside repeat-until loops; see *Do Not Use Inside Repeat-Until Loops* on page 349.

Programming

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	X1	INT

Do Not Use Inside Repeat-Until Loops



- Using an object page inside a repeat-until loop will cause a compiler error
- Components in this table cause compiler errors when used inside a repeat-until loop

Do Not Use these Components Inside a Repeat-Until Loop

Category	Component Name
Transition, Time	On Delay
	Off Delay
	Oscillator
	Pulse
	Time Base
	Measure Period
	Get Time µs
Connection	Simple Checkpoint
	Create Externally Defined Class
	Call Method of Externally Defined Class
	Advanced Checkpoint
	Advanced Checkpoint with Namespace
	Set Value
	Set Pulse
	Initialize Hardware Output
	Hardware Input Typed
	Hardware Input
	Hardware Output
	Hardware Input/Output Typed
	Hardware Input/Output
	Read Output from Hardware Typed
	Read Output from Hardware
	Non-Volatile Memory Input Typed
	Non-Volatile Memory Input/Output
	Using Old CAN and Memory Components
	Transmit CAN
	Receive CAN with Filter
	Receive CAN with ID Mask
	Receive CAN Basic
	Non-Volatile Memory Dynamic with Default
	Non-Volatile Memory Dynamic

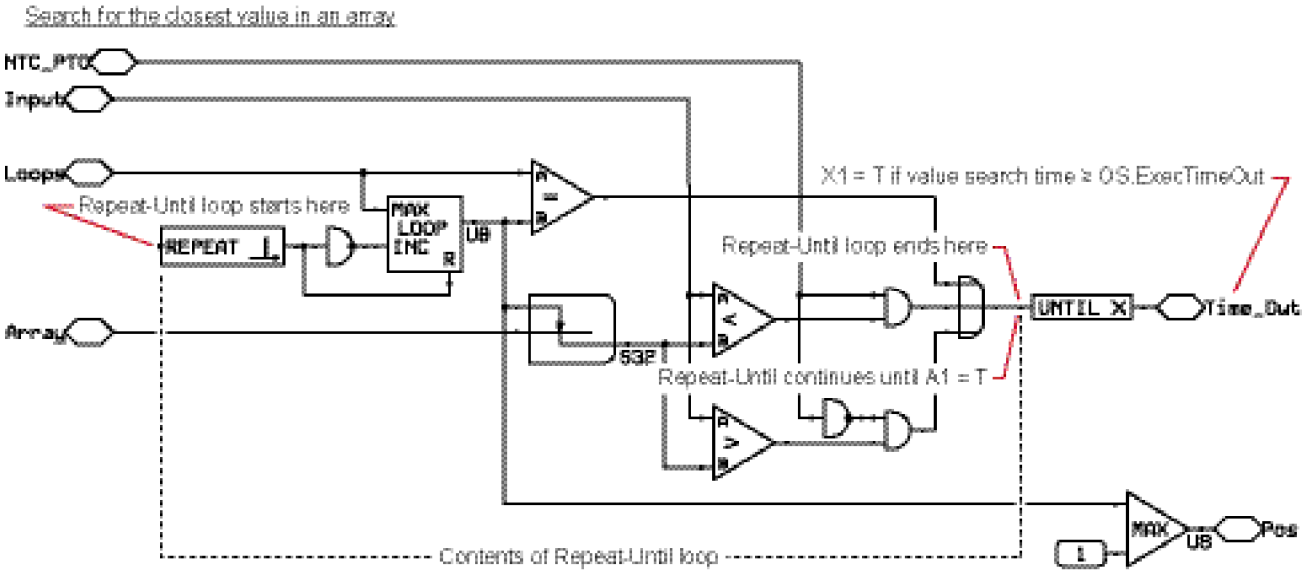
Programming

Do Not Use these Components Inside a Repeat-Until Loop (continued)

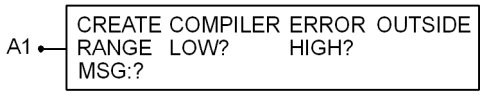
Category	Component Name
Module	Module Input
	Module Input Typed
	Module Output
	Configurable Module Connection
	Module Bus Input
	Module Bus Output
	Call Module
Access	Access App Log Enable
	Disable Raw Applog Data Readout
	Accessrights Diag Access
	Accessrights App Log Errors
	Accessrights App Log Others
Display	Define Window
	Line
	Graphic
	Text Label
	Graphic Label
	String
	String with 4 Variables
	Basic String with 4 Variables
	Basic String with 8 Variables
	Open Text Set
	Close Text Set
	Language Definition Input
	Select Language
	Define Areas Page
	Define Screen Page
Application Log	Language Definition Input
	Select Language
	Define Application Log Areas Page
	Define Application Log Page

Programming

Example—Repeat-Until



Generate Compiler Error on Range



Use:

- Aborts the compile process when the compiler finds an out-of-range constant value
- Displays a user-defined error message in the **Error/Warning/Hint Messages** tab
- Prevents a user from entering a value that compiles but causes erratic operation when downloaded

  (For example, a joystick calibration value that is too high would compile successfully but might cause erratic machine operation when downloaded)

Function:

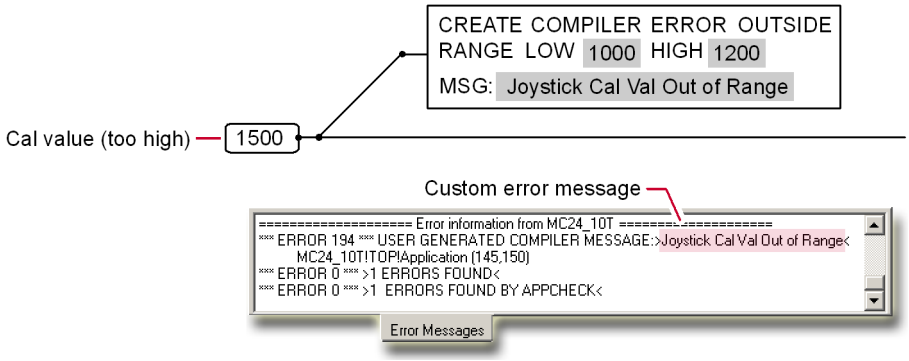
- A1 = Constant value
- **Low?**= Click with Query/Change to enter the low value for the range
- **High?**= Click with Query/Change to enter the high value for the range
- **MSG?**= Click with Query/Change to enter the error message that displays if A1 is out of range

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	—	——

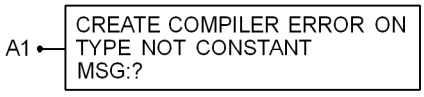
Programming

Example—Generate Compiler Error on Range



No error check will be performed if A1 is connected to a non-constant value.

Generate Compiler Error on Not Constant



Use:

- Aborts the compile process when the compiler finds variable value instead of a constant value
- Displays a user-defined error message in the **Error/Warning/Hint Messages** tab
- Prevents a user from entering a value that compiles but causes erratic operation when downloaded

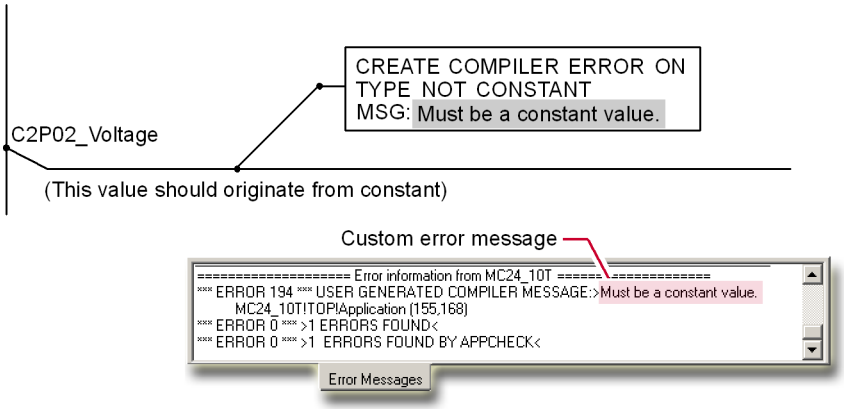
Function:

- A1 = Constant data type (any other data type aborts the compile)
- **MSG?**= Click with Query/Change to enter the error message that displays if A1 is a variable value instead of a constant value

Valid connections

Pin	Data Type	Pin	Data Type
A1	MAIN	—	——

Example—Generate Compiler Error on Not Constant



Generate Compiler Error on Type

Programming

Use:

- Aborts the compile process when the compiler finds an incorrect data type
- Displays a user-defined, custom error message in the **Error/Warning/Hint Messages** tab
- Prevents a user from entering a value that compiles but causes erratic operation when downloaded

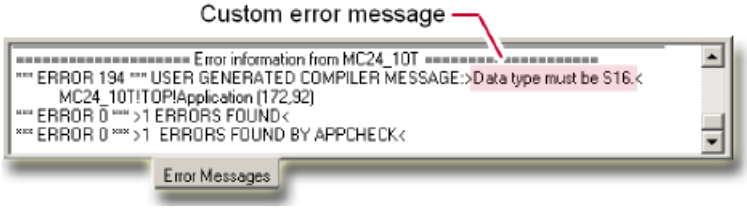
Function:

- A1 = Data type
- **TYPE DIFFERS FROM?**= Click with Query/Change to enter the allowed data type (any other data type aborts compile)
- **MSG?**= Click with Query/Change to enter the error message that displays if A1 is an incorrect data type

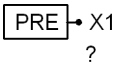
Valid connections

Pin	Data Type	Pin	Data Type
A1	MAIN	—	——

Example—Generate Compiler Error on Type



Predefine Type



Use:

- Defines the data type of a signal that does not have a data type
  - Needed to define the data type of a **Module Input** signal routed to a hardware output
  - Needed to define the data type of a **Module Output** signal routed to a **Checkpoint** component
- Force connecting the correct signal to an input or output
  - A signal that you route to a predefined signal must either have no data type or a matching data type; otherwise you get a compiler error

Function: X1 = Outputs the **?**data type

Valid connections

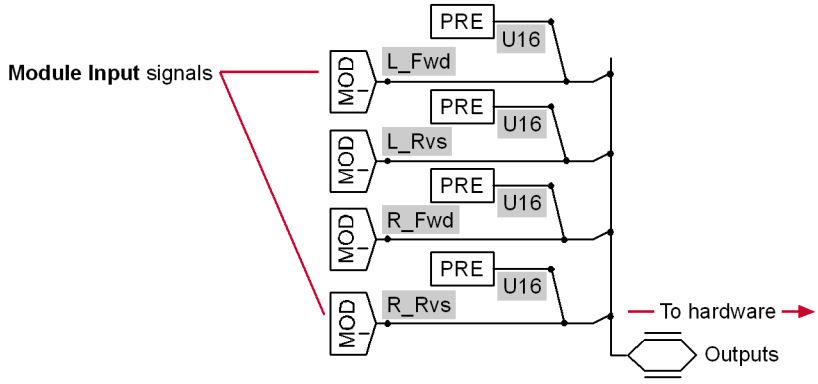
Pin	Data Type	Pin	Data Type
—	——	X1	MAIN, STRING

See also *Example 1—Call Module* on page 345 for another example of how to use this component.

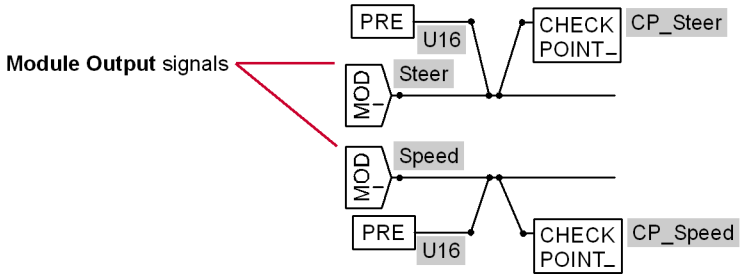
Programming

Example—Predefine Type

Define the data type of **Module Input** signals routed to hardware



Define the data type of **Module Output** signals routed to **Checkpoints**



SIL2 Certified

Certification of a market-released version of the PLUS+1 GUIDE software may occur after its initial release. Contact Danfoss for the certification status of your version of the PLUS+1 GUIDE software.

Use: Enable critical safety-related functions in an application

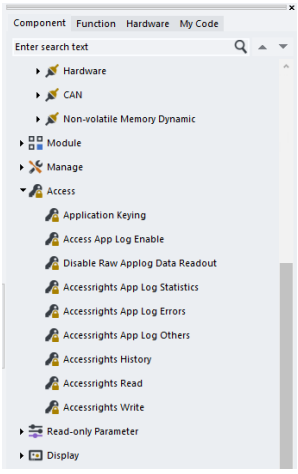
Function:

- X1 = T if all of the following conditions are met:

Programming

- The PLUS+1 program that you are using must both be market-released and certified to comply with IEC 61508 requirements.
- The *.HWD file that you are using must be certified to comply with IEC 61508 requirements.
- You clicked **Yes** in a dialog box that opened during the compile process to certify that your application fulfills IEC 61508 requirements.

Access Menu



Application Keying

Application Keying TARGET HARDWARE PART NUMBER PARTNUMBER

Use:

- Limit the download of an application to hardware that is sold by a specific distributor
- Hardware must have the application key feature

Function:

- Danfoss embeds an application key value in hardware that has the application key feature
- Replace the **Application Keying** component's **PARTNUMBER** value with the application key value that is embedded in the keyed hardware
- To compile the application, the **Inspector** tab's **Target part number** value must match the **Application Keying** component's **PARTNUMBER** value

  The compile process embeds the application key value in the LHX file
- To download the application, the application key value that is embedded in the LHX file must match the application key value that is embedded in the keyed hardware
- If a customer:
  - Receives only the LHX file, the distributor does not have to hide the **Application Keying** component in the PLUS+1 application
  - Receives the PLUS+1 application, the distributor should put the **Application Keying** component in a locked page that:

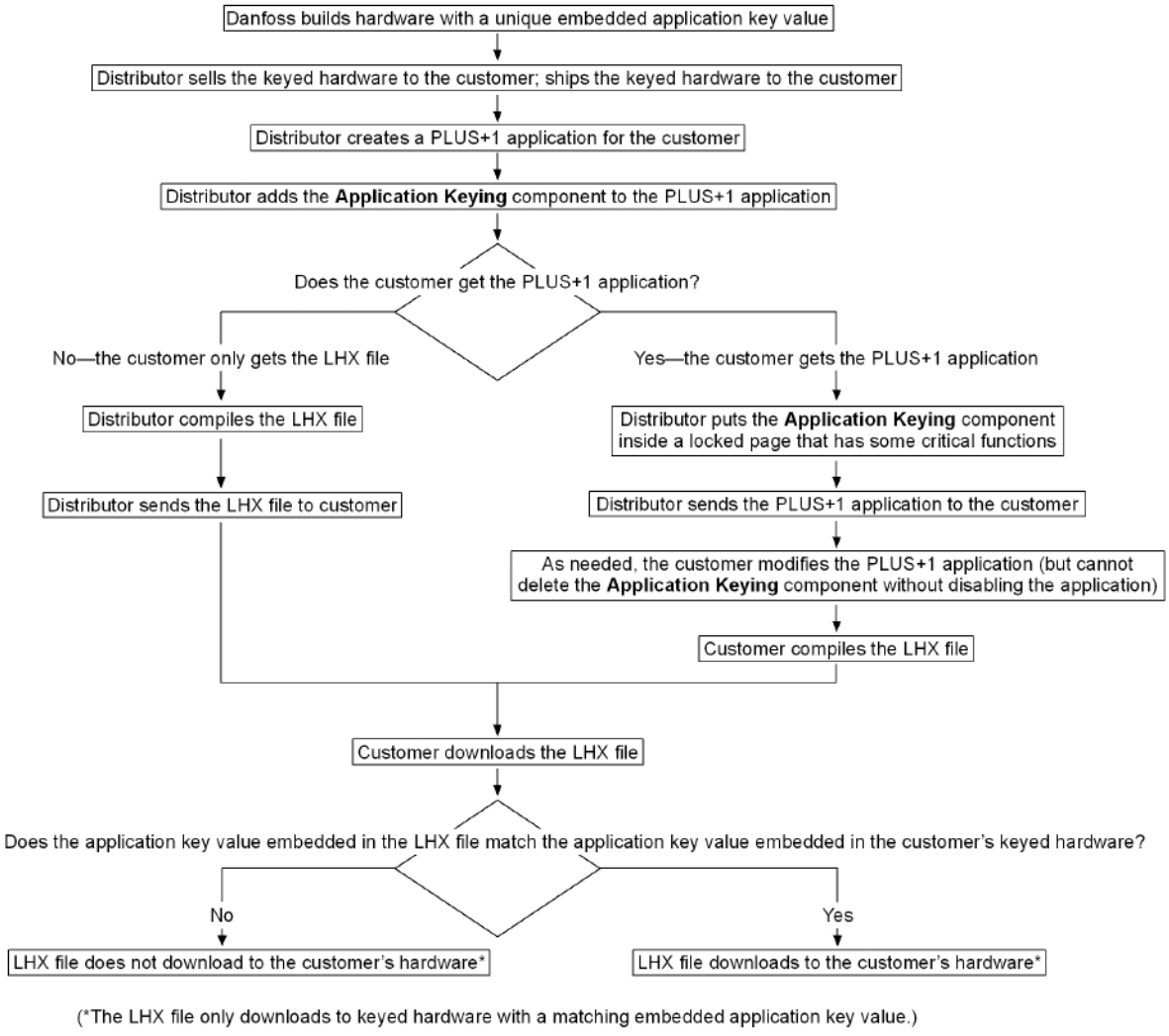
    Contains critical functions that cannot be easily duplicated by the customer

    Is isolated from items (such as function blocks and configuration settings) that the customer may need to modify

Programming

Application Keying — Example 1

This flowchart shows an example of how a distributor can use the **Application Keying** component along with keyed hardware to limit the downloads of an application that they have developed to only the keyed hardware that they have sold.

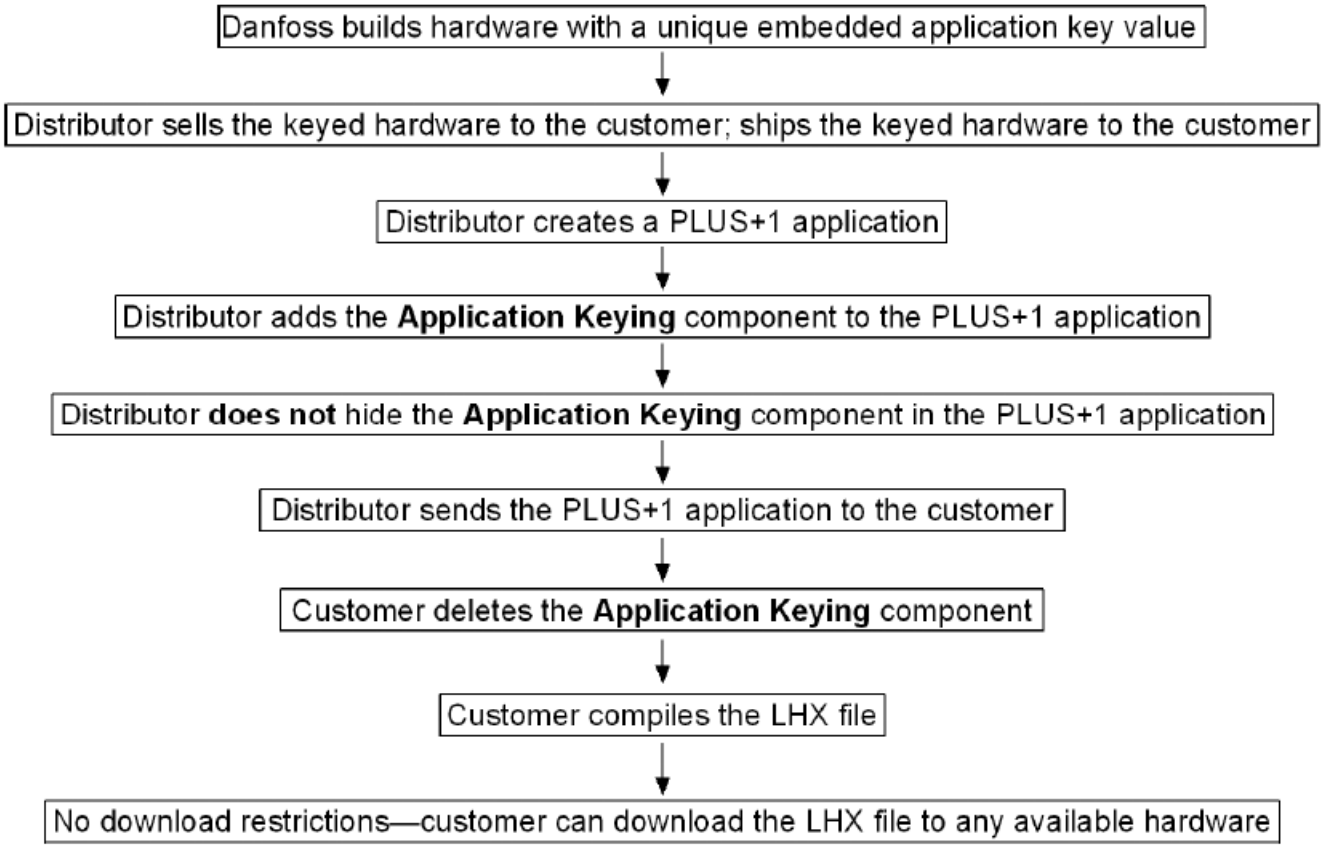


(*The LHX file only downloads to keyed hardware with a matching embedded application key value.)

Application Keying — Example 2

This flowchart shows an example of how a customer can defeat the hardware keying feature if a distributor does not hide the **Application Keying** component in the PLUS+1 application that they develop.

Programming



Access App Log Enable

Use:

- Limits access by fully licensed versions of the PLUS+1 Service Tool program to an application data log
- Required to enable access by the free version of the PLUS+1 Service Tool program to an application data log
- Without this component the free version of the PLUS+1 Service Tool program cannot access an application data log

Function:

- If:
  - A1 = True, the Service Tool program can access the application data log
  - A1 = False, the Service Tool program cannot access the application data log

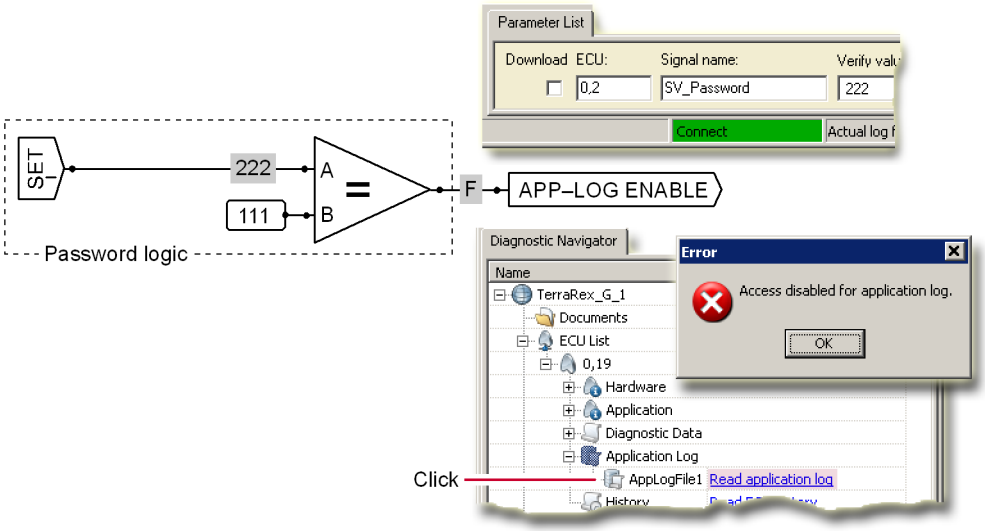
Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	—	——

Programming

Example—Access App Log Enable

Wrong password/no password when using the free PLUS+1 Service Tool program



Disable Raw Applog Data Readout

Use:

- Works with password logic to limit the PLUS+1 Service Tool program's access to the P1A file that contains raw data from the application data log

Function:

- If:
  - A1 = True, the Service Tool program cannot access the P1A file

    The Service Tool program can only write a CSV file that contains data allowed by other **Access** components, such as the **App-Log Error Access** component
  - A1 = False, the Service Tool program can access the P1A file

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	—	——

Accessrights App Log Statistics

Use:

- Works with password logic to limit the PLUS+1 Service Tool program's access to statistical data contained in an application data log
- Data written to an application data log has a user-defined:
  - **S**, **E**, or **O** letter **Tag** that categorizes the data as statistical, error, or "other" data
  - **Access** level from **0**–**9**

Function:

- A1 can range from 0–9
- The Service Tool program:

Programming

  - Can access statistical data in the application data log that has an access level ≥ A1
  - Cannot access statistical data in the application data log that has an access level < A1
- Can be placed anywhere in an application

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	—	——

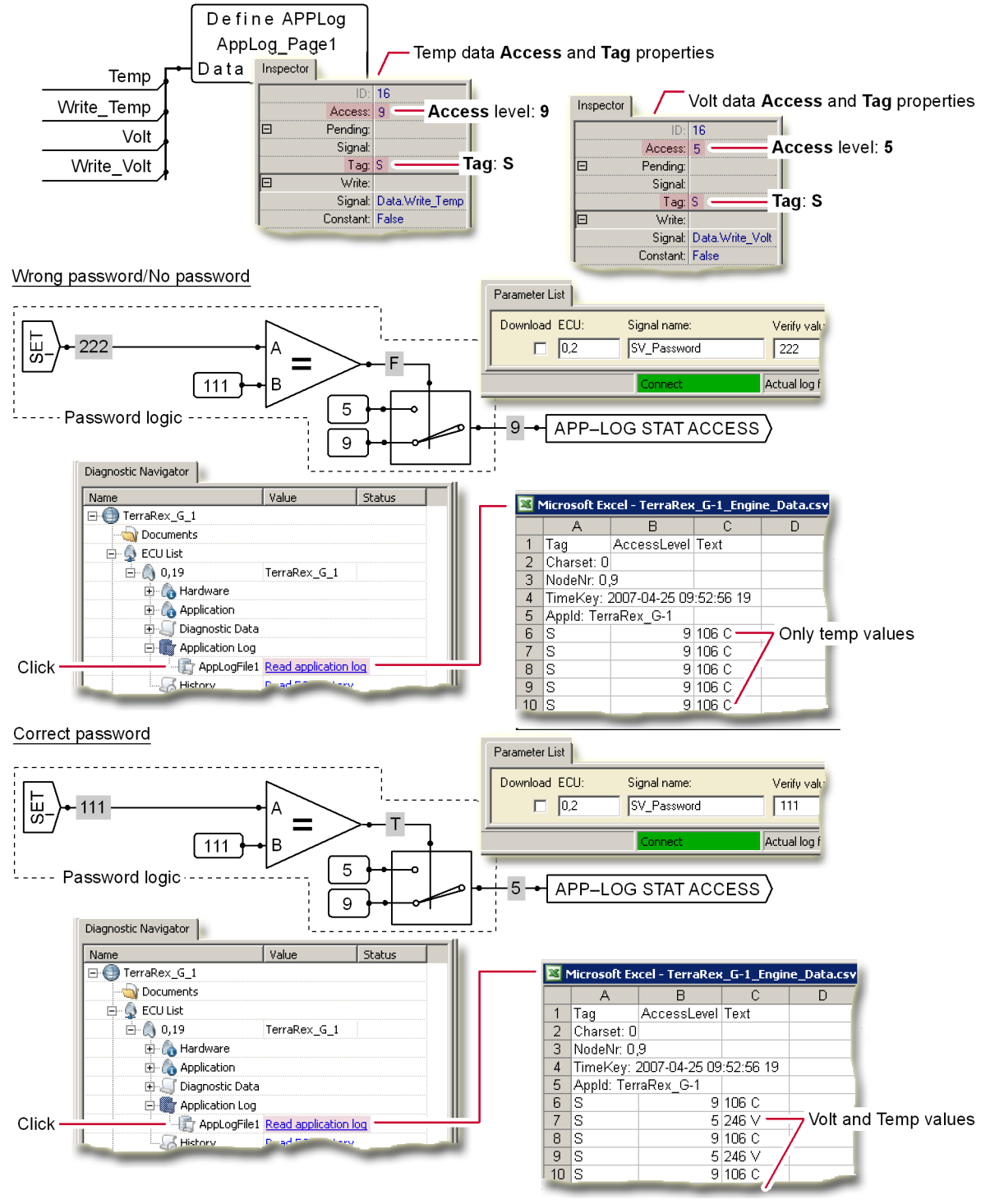
Diagnostic Data Access levels

A1 Access Value	Can Access Statistical Data with these access levels										
0 APP-LOG STAT ACCESS	0	1	2	3	4	5	6	7	8	9	Most
3 APP-LOG STAT ACCESS				3	4	5	6	7	8	9	
6 APP-LOG STAT ACCESS							6	7	8	9	
9 APP-LOG STAT ACCESS										9	Least

Programming

Example—Accessrights App Log Statistics

Inspector tab inside **Define AppLog** component



Accessrights App Log Errors

A1 — APP–LOG ERROR ACCESS

Use:

Programming

- Works with password logic to limit the PLUS+1 Service Tool program's access to error data contained in an application data log
- Data written to an application data log has a user-defined:
  - **S**, **E**, or **O** letter **Tag** that categorizes the data as statistical, error, or "other" data
  - **Access** level from **0**–**9**

Function:

- A1 can range from 0–9
- The Service Tool program:
  - Can access error data in the application data log that has an access level ≥ A1
  - Cannot access error data in the application data log that has an access level < A1
- Can be placed anywhere in an application

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	—	——

Error Data Access Levels

A1 Access Level	Can Access Error Data with These Access Levels										
0 APP-LOG ERROR ACCESS	0	1	2	3	4	5	6	7	8	9	Most
3 APP-LOG ERROR ACCESS				3	4	5	6	7	8	9	
6 APP-LOG ERROR ACCESS							6	7	8	9	
9 APP-LOG ERROR ACCESS										9	Least

The APP-LOG STATS ACCESS component works like the APP-LOG ERROR ACCESS component to limit access to types of data contained in the application data log.

See *Example—Accessrights App Log Statistics* on page 360.

Accessrights App Log Others

A1 APP–LOG OTHERS ACCESS

Use:

- Works with password logic to limit the PLUS+1 Service Tool program's access to "other" data contained in an application data log
- Data written to an application data log has a user-defined:
  - **S**, **E**, or **O** letter **Tag** that categorizes the data as statistical, error, or other data
  - **Access** level from **0**–**9**

Function:

- A1 can range from 0–9
- The Service Tool program:
  - Can access other data in the application data log that has an access level ≥ A1
  - Cannot access other data in the application data log that has an access level < A1
- Can be placed anywhere in an application

Programming

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	—	——

Other Data Access Levels

A1 Access Value	Can Access Other Data with These Access Levels										
0 APP-LOG OTHERS ACCESS	0	1	2	3	4	5	6	7	8	9	Most
3 APP-LOG OTHERS ACCESS				3	4	5	6	7	8	9	
6 APP-LOG OTHERS ACCESS							6	7	8	9	
9 APP-LOG OTHERS ACCESS										9	Least

The **APP-LOG STATS ACCESS** component works like the **APP-LOG OTHERS ACCESS** component to limit access to types of data contained in the application data log. See *Example—Accessrights App Log Statistics* on page 360.

Accessrights History

A1 HISTORY ACCESS

Use:

- Works with password logic to limit the PLUS+1 Service Tool program's access to a unit's history

Function:

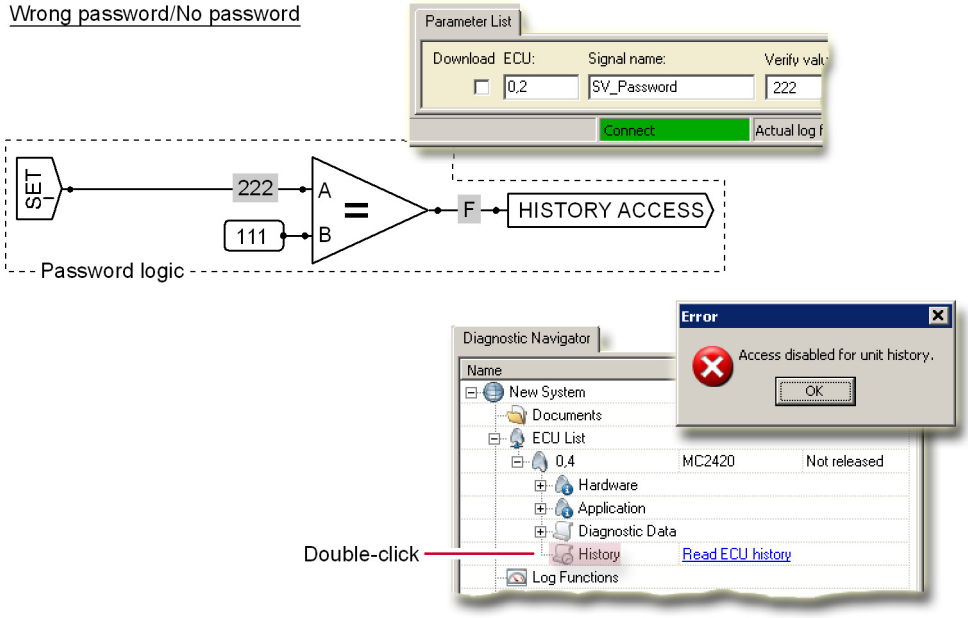
- If:
  - A1 = False, the Service Tool program cannot read the unit history
  - A1 = True, the Service Tool program can read the unit history
- Can be placed anywhere in an application
- Overrides any access right set in the Service Tool program

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	——	——

Programming

Example—Accessrights History



Accessrights Read

A1 • READ ACCESS

Use:

- Works with password logic to limit the PLUS+1 Service Tool program's ability to read component values
  - Some PLUS+1 components—such as the **Checkpoint** and **Memory** components—have a read access level feature; their read access levels can range from 0 to 9
  - The **A1** value applied to the **Read Access** component works with individual components' read access levels to limit the values that the Service Tool program can read without the correct password

Function:

- A1 can range from 0–9
- Using the Service Tool program, a user:
  - Can read the value of any component whose read access level ≥ A1
  - Cannot read the value of any component whose read access level < A1
- Sets read access levels for all components in an application
- Can be placed anywhere in an application
- Overrides any read access level set in the Service Tool program

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	—	——

Programming

Read Access Levels

A1 Value*	Can Read Values from Components with These Read Access Levels										
0 READ ACCESS	0	1	2	3	4	5	6	7	8	9	Most
3 READ ACCESS				3	4	5	6	7	8	9	
6 READ ACCESS							6	7	8	9	
9 READ ACCESS										9	Least

* Typically set with password logic.

Accessrights Write

A1 WRITE ACCESS

Use:

Works with password logic to limit the PLUS+1 Service Tool program's ability to write values to components:

- Some PLUS+1 components—such as **Memory** components—have a write access level feature; their write access levels can range from 0 to 9
- The **A1** value applied to the **Write Access** component works with individual components' write access levels to limit the values that the Service Tool program can change (write to) without the correct password

Function:

- A1 can range from 0–9
- With the Service Tool program, a user:
  - Can write a value to any component whose write access level ≥ A1
  - Cannot write a value to any component whose write access level < A1
- Sets write access levels for all components in an application
- Can be placed anywhere in an application
- Overrides any write access level set in the Service Tool program

Valid connections

Pin	Data Type	Pin	Data Type
A1	INT	—	——

Write Access Levels

A1 Value*	Can Write Values to Components with These Write Access Levels										
0 WRITE ACCESS	0	1	2	3	4	5	6	7	8	9	Most
3 WRITE ACCESS				3	4	5	6	7	8	9	
6 WRITE ACCESS							6	7	8	9	
9 WRITE ACCESS										9	Least

* Typically set with password logic.

Read Access and Write Access — Example 1

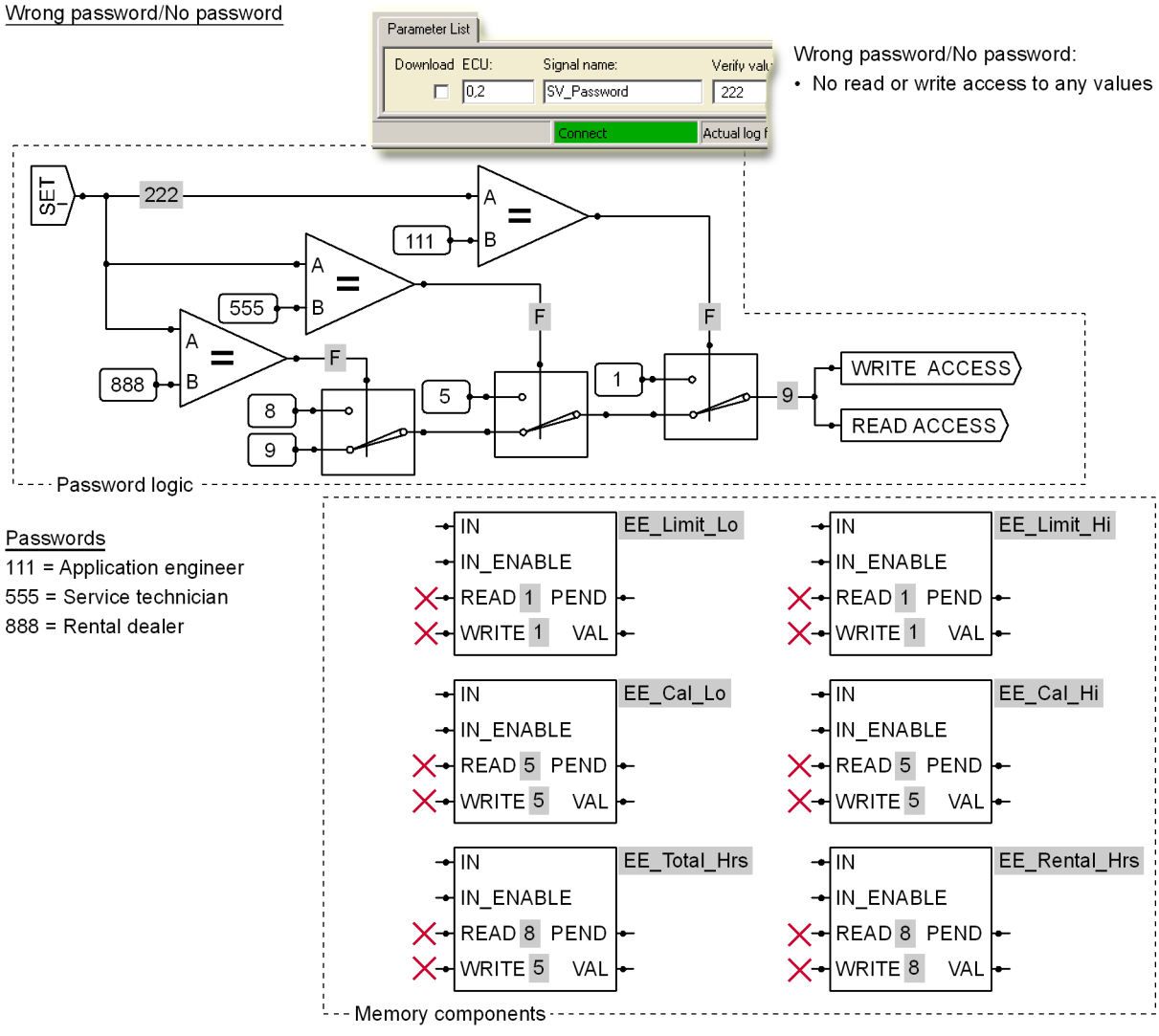
The examples on this and on the following pages show how password logic used with the **Read Access** and **Write Access** components can limit the PLUS+1 Service Tool program's ability to read and write component values.

Programming

In the example below, someone using the Service Tool program enters either the wrong password or no password at all.

The password logic applies a value of **9** to the **Write Access** and **Read Access** components. The Service Tool program can only read and write to **Memory** components with read and write access levels of **9**. (No components here have an access level of **9**).

A user with no password or the wrong password cannot view or change any values.



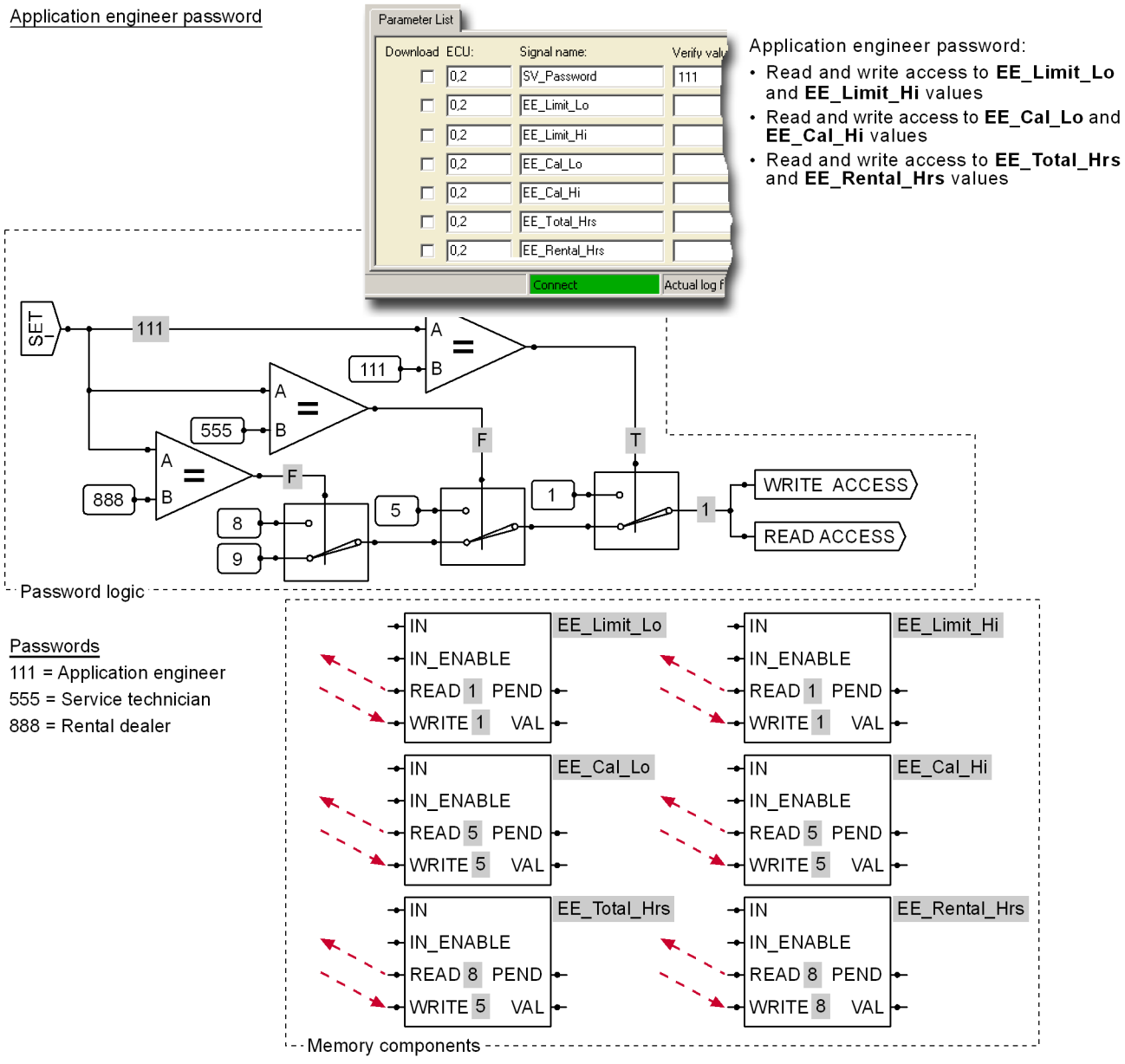
Read Access and Write Access — Example 2

The example below shows what happens when an application engineer uses the PLUS+1® Service Tool to enter a password of **111**.

The password logic applies a value of **1** to the **Write Access** and **Read Access** components. A value of **1** lets the PLUS+1® Service Tool program read and write to all **Memory** components with read and write access levels equal to or greater than **1**.

The application engineer can view and change all values.

Programming



Read Access and Write Access — Example 3

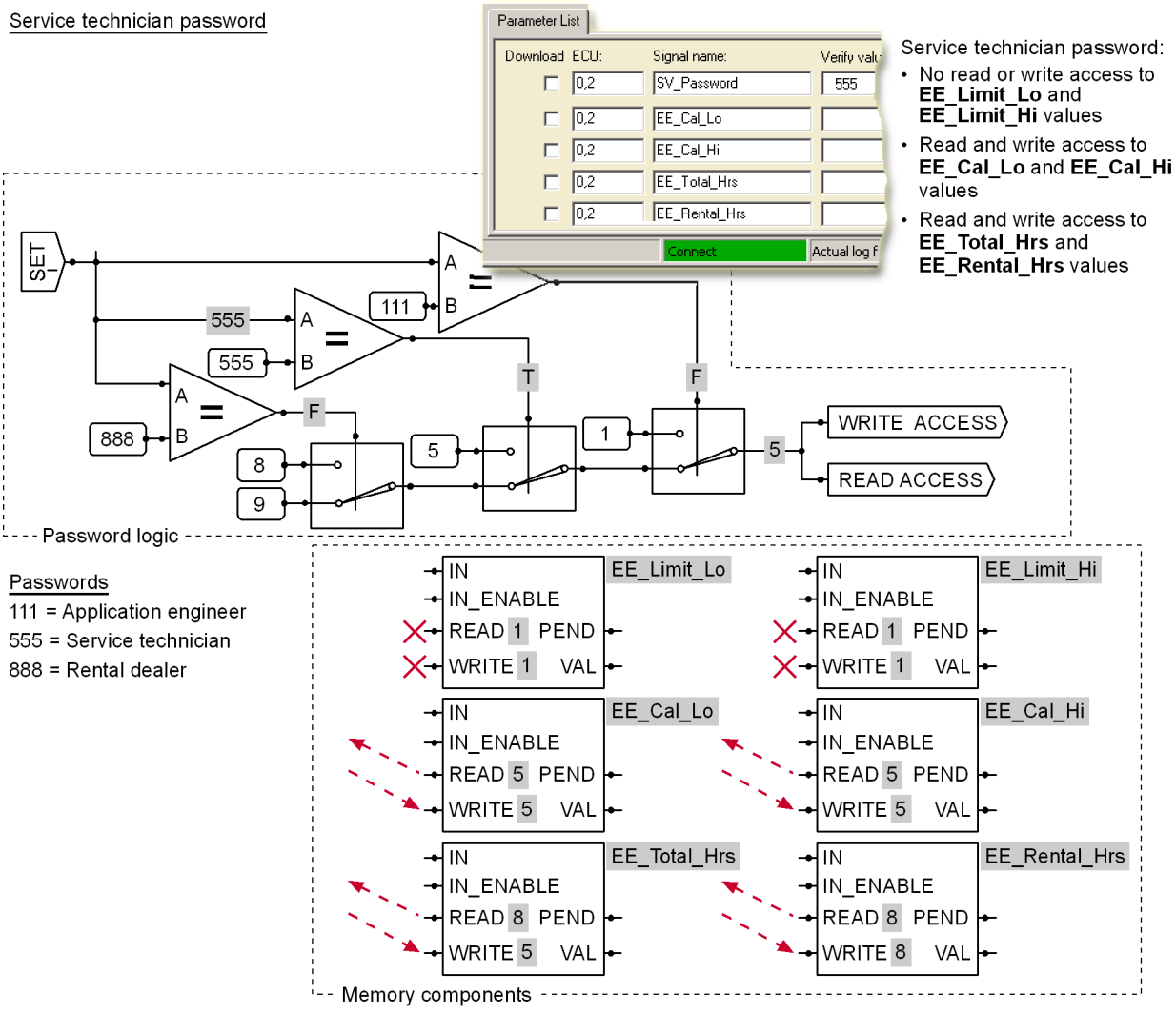
The example below shows what happens when a service technician uses the PLUS+1® Service Tool to enter a password of **555**.

The password logic applies a value of **5** to the **Write Access** and **Read Access** components. A value of **5** lets the PLUS+1® Service Tool program read and write to all **Memory** components with read and write access levels equal to or greater than **5**.

The service technician:

- cannot view or change limits.
- can view and change calibration values.
- can view and change (reset) total run hours.
- can view and change (reset) rental hours.

Programming



Read Access and Write Access — Example 4

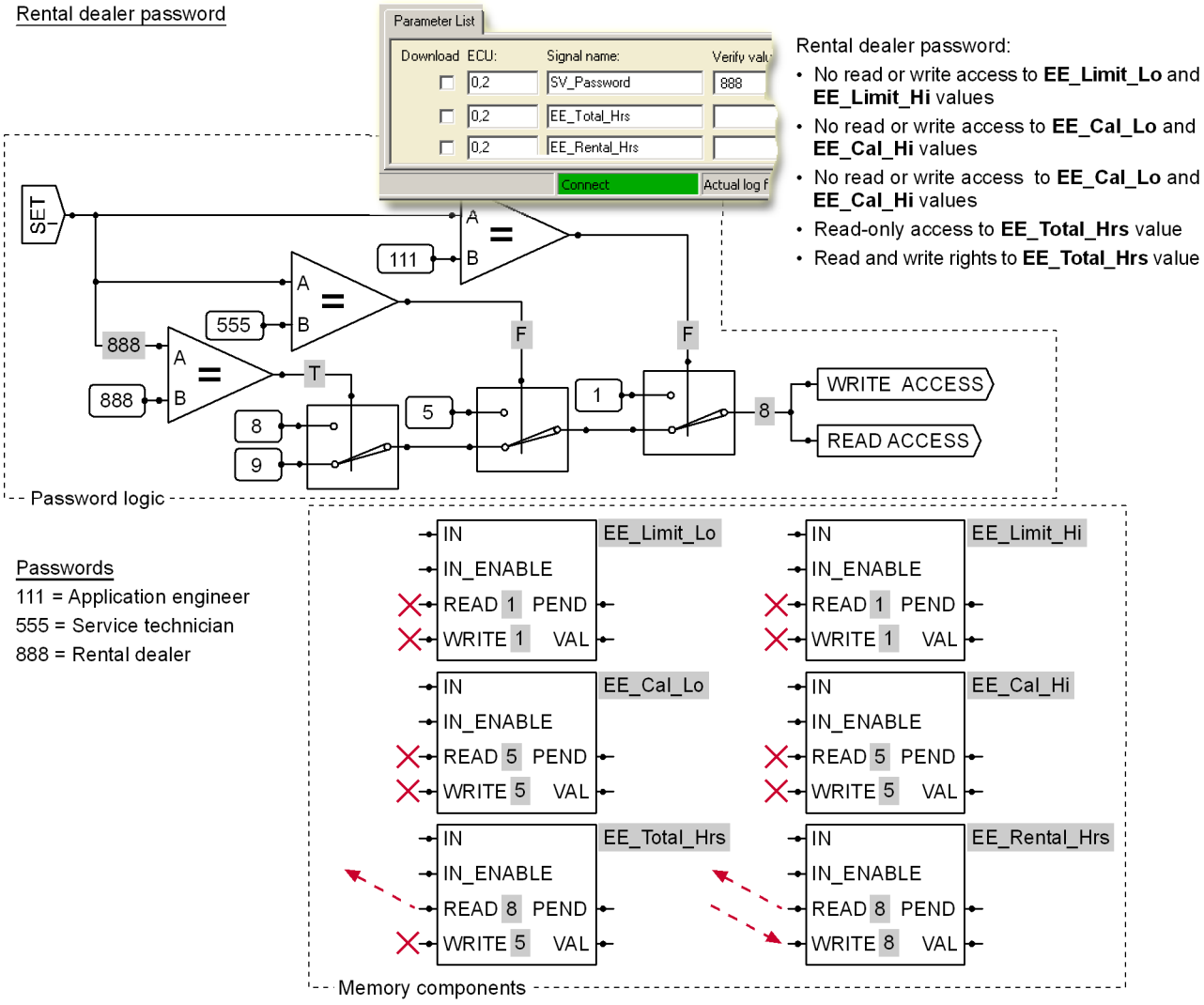
The example below shows what happens when a rental dealer uses the PLUS+1 Service Tool program to enter a password of **888**.

The password logic applies a value of **8** to the **Write Access** and **Read Access** components. A value of **8** lets the Service Tool program read and write to all **Memory** components with read and write access levels equal to or greater than **8**.

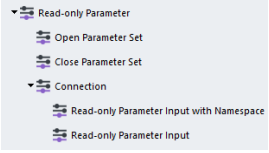
The rental dealer:

- cannot view or change limits.
- cannot view or change calibration values.
- can view but not change (reset) total run hours.
- can view and change (reset) rental hours.

Programming



Read-only Parameter Menu



Open Parameter Set

Use: Marks the start of a set of read-only parameter inputs.

Function:

- X1 = True if the parameters in a read-only parameters lhx file matches the parameters in the application

Programming

  - **TYPE** name, parameter names, parameter order, and data types must match
  - Program the application so that a True output from this component writes values in the read-only parameters lhx file to the controller
- X1 = False if the parameters in a read-only parameters lhx file do not match the parameters in the application
  - Program the application to deal with a False output from this component
- **TYPE** = Replace with a user-friendly name for the parameter set

See *Example—Read-only Parameter Input* on page 371.

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	BOOL

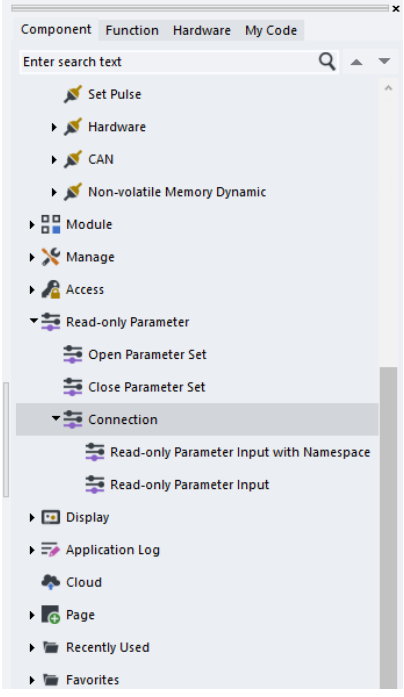
Close Parameter Set

Use: Marks the end of a set of read-only parameter inputs.

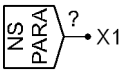
Function: **COMMENT** = Replace with a useful comment

See *Example—Read-only Parameter Input* on page 371.

Connection



Read-only Parameter Input with Namespace



Use:

Programming

- Inputs a read-only parameter to an application
- When copying a parameter set, you only have to change the page's **Namespace**.

  You do not have to change individual **?** parameter names.

Function:

- Prefixes the page's **Namespace** to the **?** parameter name
- **?** = Name of the parameter
- X1 = Parameter value input from a read-only parameters lhx file

See *Example—Read-only Parameter Input with Namespace* on page 370.

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	INT, BOOL, ARRAY

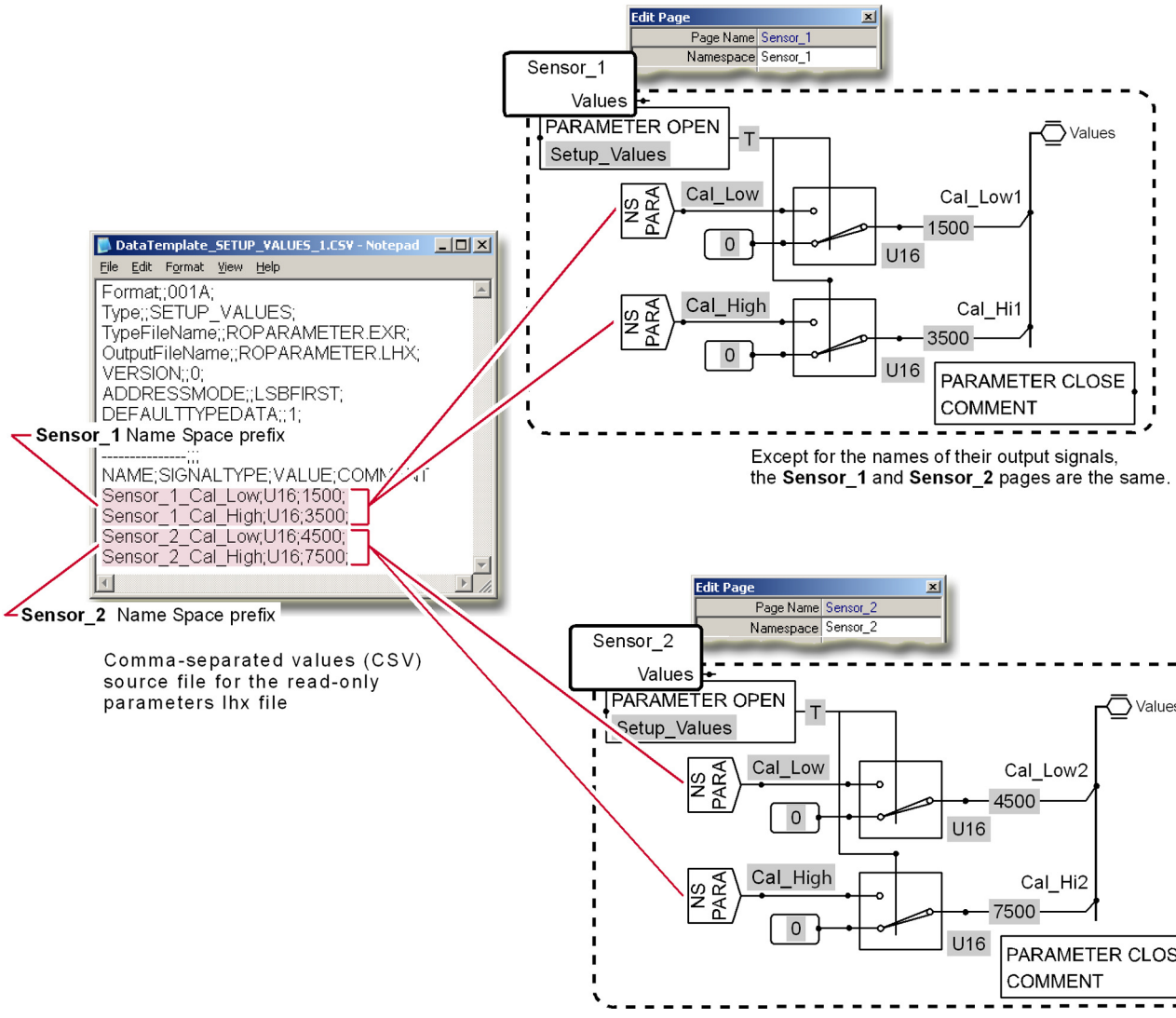
Example—Read-only Parameter Input with Namespace

Danfoss recommends that you use Notepad instead of Excel to edit and save your parameters source file. Excel can produce unpredictable results.

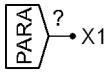
The following example shows:

- The source file for the read-only parameters lhx file as it appears in Notepad.
- That each parameter name in the source file has a **Namespace** prefix.

Programming



Read-only Parameter Input



Use: Inputs a read-only parameter to an application.

Function:

- **?** = Name of the parameter
- X1 = Parameter value input from a read-only parameters lhx file

See *Example—Read-only Parameter Input* on page 371.

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	INT, BOOL, ARRAY

Example—Read-only Parameter Input

Programming

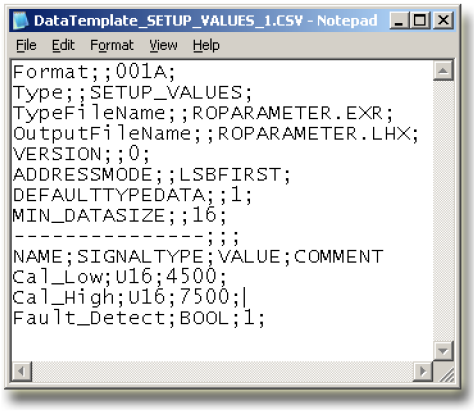
Danfoss recommends that you use Notepad instead of Excel to edit and save your parameters source file. Excel can produce unpredictable results.

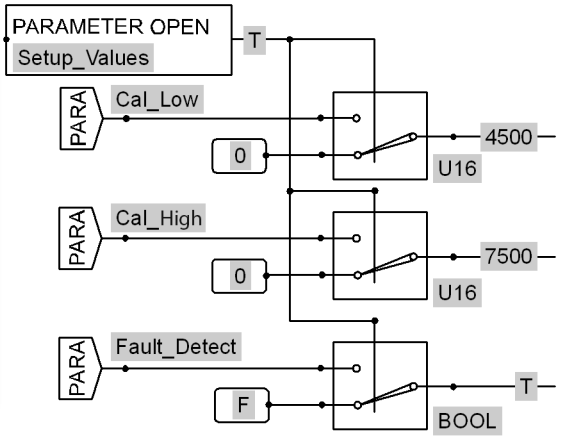
The following example shows:

- The source files for read-only parameters lhx files as they appear in Notepad.
- That the **PARAMETER OPEN**component outputs a True for a valid read-only parameters LHX file and outputs a False for an invalid read-only parameters lhx file.
- How True inputs to the three **Switch 2** components cause these three components to apply the read-only parameter values of 4500, 7500, and True.
- How False inputs to the three **Switch 2** components cause these three components to apply the internal values of 0, 0, and False.

Comma-separated values (CSV) source file for the read-only parameters lhx file

Names—OK
Data types—OK
Order—OK

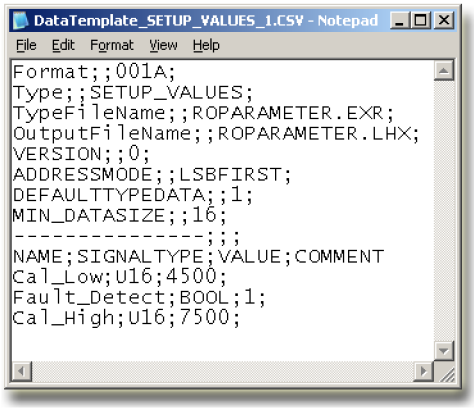


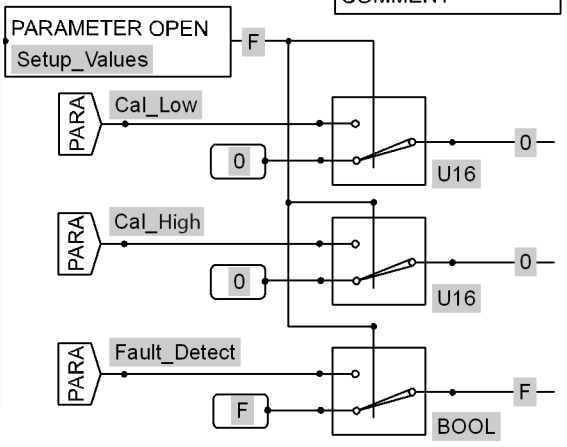


If the read-only parameter lhx file is valid, then the **PARAMETER OPEN** component X1 output = T

PARAMETER CLOSE
COMMENT

Names—OK
Data types—OK
Order—Not OK



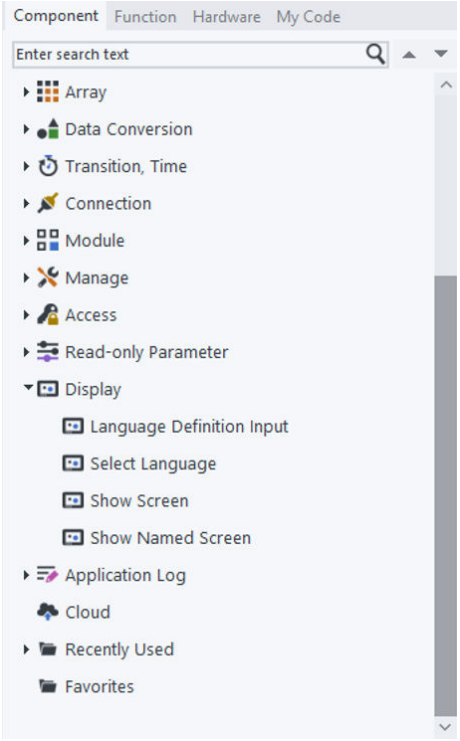


If the read-only parameter LHX file is not valid, then the **PARAMETER OPEN** component's X1 output = F

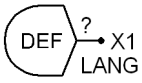
PARAMETER CLOSE
COMMENT

Programming

Display Menu



Language Definition Input



Use: Combine with the **Select Language** component to enable switching between the languages displayed in a PLUS+1 graphical terminal. *Example—Select Language/Language Definition* on page 374 shows how to use this component with the **Select Language** component

Function: **?** = Language as defined in the **Define Screen** page's **Screen Library** tab.

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	LANG

Select Language

Use: Combine with the **Language Definition Input** component to enable switching between the languages displayed in a PLUS+1® graphical terminal. The following example shows how to use this component with the **Language Definition Input** component

Function:

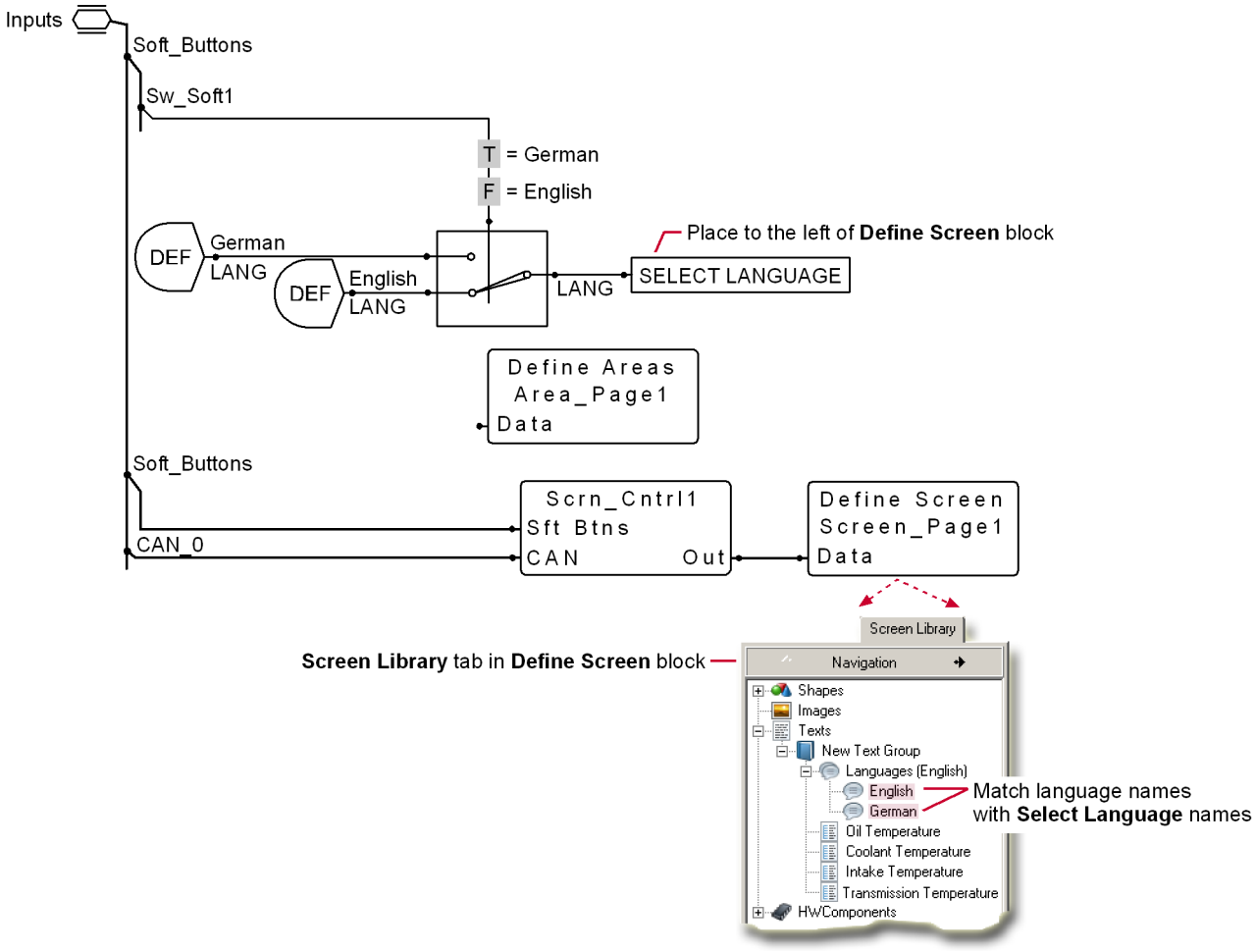
Programming

- A1 = Language as defined in the **Define Screen** page's **Screen Library** tab
- Must be placed to the left of the **Define Screen** or **Define Applog** page to work

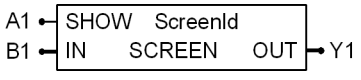
Valid connections

Pin	Data Type	Pin	Data Type
A1	LANG	—	——

Example—Select Language/Language Definition



Show Screen



Use: Instantiates a **Screen Definition**.

Function:

- **ScreenId** = Replace with the name of the instantiated **Screen Definition**
- If:
  - A1 = True, enable the display of the instantiated Screen Definition
  - A1 = False, disable the display of the instantiated Screen Definition
- B1 = Bus connection for signals from the application to the instantiated Screen Definition
- Y1 = Bus connection for signals from the instantiated Screen Definition to the application

Programming

This is a hardware-dependent component for use with specific models of graphical terminals. This component only becomes available when you install the hardware (HWD) file for a supported graphical terminal in the **Project Manager** tab.

For more about how to use this component, refer to the *Vector-Based Screen Editor* on page 450 chapter.

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	Y1	U8, U16, U32, S8, S16, S32, BOOL, COLOR
B1	U8, U16, U32, S8, S16, S32, BOOL, COLOR	——	——

Show Named Screen

Use: Instantiates a **Screen Definition**.

- Should be used instead of the **Show Screen** component when you want to include Diagnostic and Non-volatile parameters of called POUs in the diagnostic data.

Function:

- **InstanceName** = Enter the name of the instance
- **ScreenId** = Replace with the name of the instantiated **Screen Definition**
- If:
  - A1 = True, enable the display of the instantiated Screen Definition
  - A1 = False, disable the display of the instantiated Screen Definition
- B1 = Bus connection for signals from the application to the instantiated Screen Definition
- Y1 = Bus connection for signals from the instantiated Screen Definition to the application

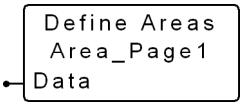
This is a hardware-dependent component for use with specific models of graphical terminals. This component only becomes available when you install the hardware (HWD) file for a supported graphical terminal in the **Project Manager** tab.

For more about how to use this component, refer to the *Vector-Based Screen Editor* on page 450 chapter.

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	Y1	U8, U16, U32, S8, S16, S32, BOOL, COLOR
B1	U8, U16, U32, S8, S16, S32, BOOL, COLOR	——	——

Define Areas Page



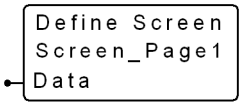
This is a hardware-dependent component for use with specific models of graphical terminals. This component only becomes available when you install the hardware (HWD) file for a supported graphical terminal in **Project Manager** tab.

The **File menu's** > **Import Page** > **Import Block** commands do not work with this component. Importing a **Define Areas** page strips the page of its contents.

Programming

For more about how to use this component, refer to *Classic Screen Editor* on page 428.

Define Screen Page

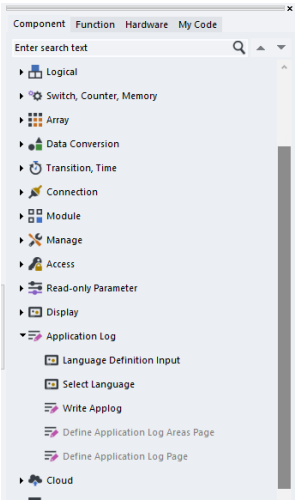


This is a hardware-dependent component for use with specific models of graphical terminals. This component only becomes available when you install the hardware (HWD) file for a supported graphical terminal in **Project Manager** tab.

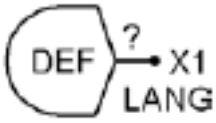
The **File menu's** > **Import Page** > **Import Block** commands do not work with this component. Importing a **Define Areas** page strips the page of its contents.

For more about how to use this component, refer to the *Classic Screen Editor* on page 428.

Application Log Menu



Language Definition Input



Use: Combine with the **Select Language** component to be able to write data to an application data log in different languages. The *Example—Language Definition/Select Language* on page 377 shows how to use this component with the **Select Language** component.

Function: **?** = Language as defined in the **Define APPLog** page's **Screen Library** tab.

Valid connections

Pin	Data Type	Pin	Data Type
—	——	X1	LANG

Select Language

Use: Combine with the **Lang** component to be able to write data to an application data log in different languages.

Programming

The *Example—Language Definition/Select Language* on page 377 shows how to use this component with the **Language Definition Input** component.

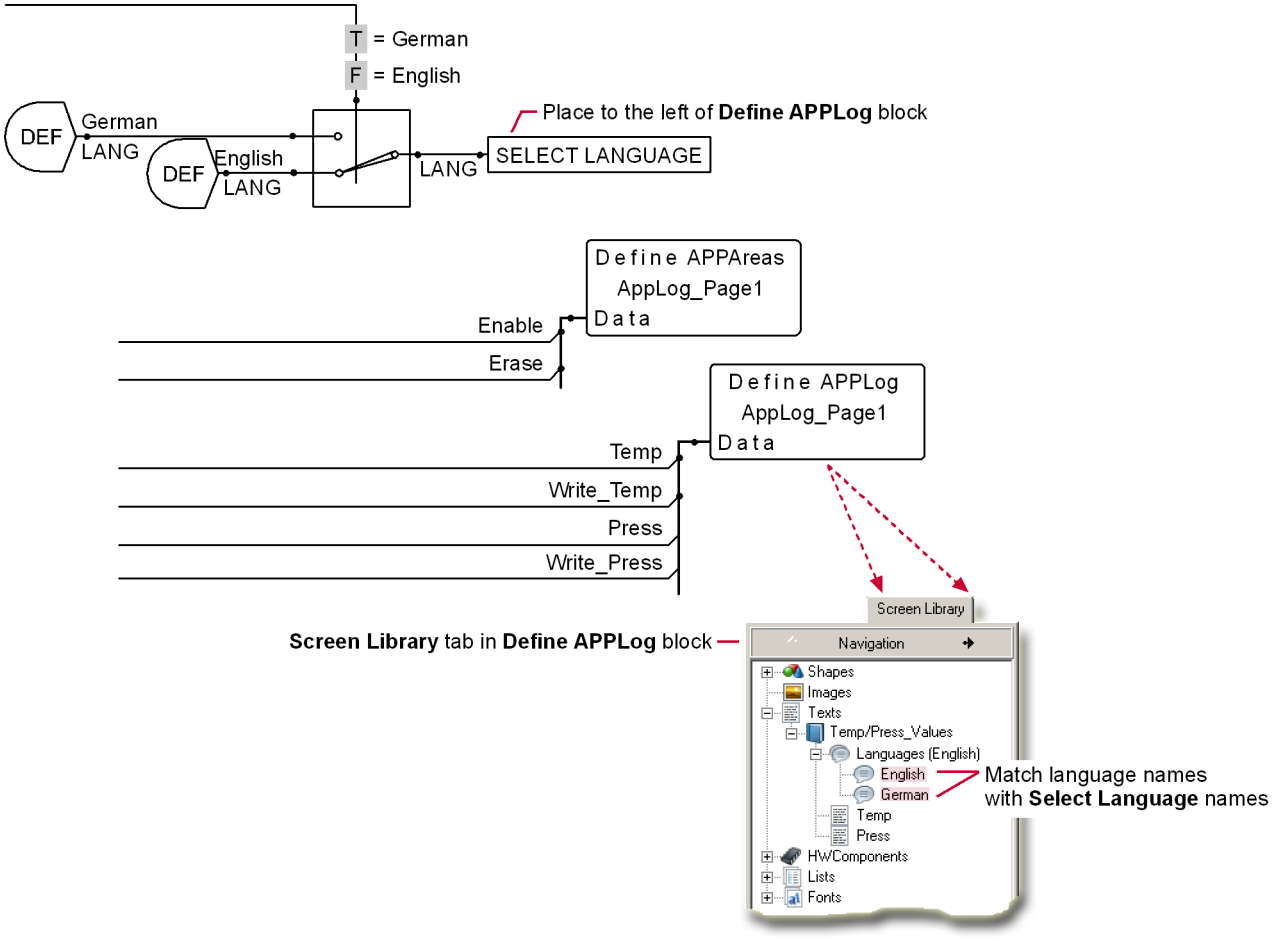
Function:

- A1 = Language as defined in the **Define APPLog** page's **Screen Editor** tab.
- Must be placed to the left of the **Define Screen** or **Define Applog** page to work.

Valid connections

Pin	Data Type	Pin	Data Type
A1	LANG	—	——

Example—Language Definition/Select Language



Write Applog

Use: Instantiates an **ApplogDef** (Application Log Definition):

- Defines what data gets logged
- Sets the size of the application log

Function:

- **ApplogDef** = Replace with the name of the instantiated **ApplogDef**
- If:

Programming

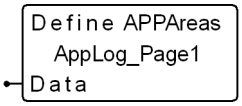
  - A1 = True, enable the instantiated **ApplogDef** (A1 must be True to log data)
  - A1 = False, disable the instantiated **ApplogDef**
- B1 = Bus connection for data signals from the application to the instantiated **ApplogDef**
- Y1 = Bus connection for data signals from the instantiated **ApplogDef**

This is a hardware-dependent component for use with specific PLUS+1 hardware. This component only becomes available when you install the hardware (HWD) file for supported hardware in the **Project Manager** tab.

Valid connections

Pin	Data Type	Pin	Data Type
A1	BOOL	Y1	U8, U16, U32, S8, S16, S32, BOOL
B1	U8, U16, U32, S8, S16, S32, BOOL	——	——

Define Application Log Areas Page



Use: Implements application data logging; use with the **Define APPLog** component.

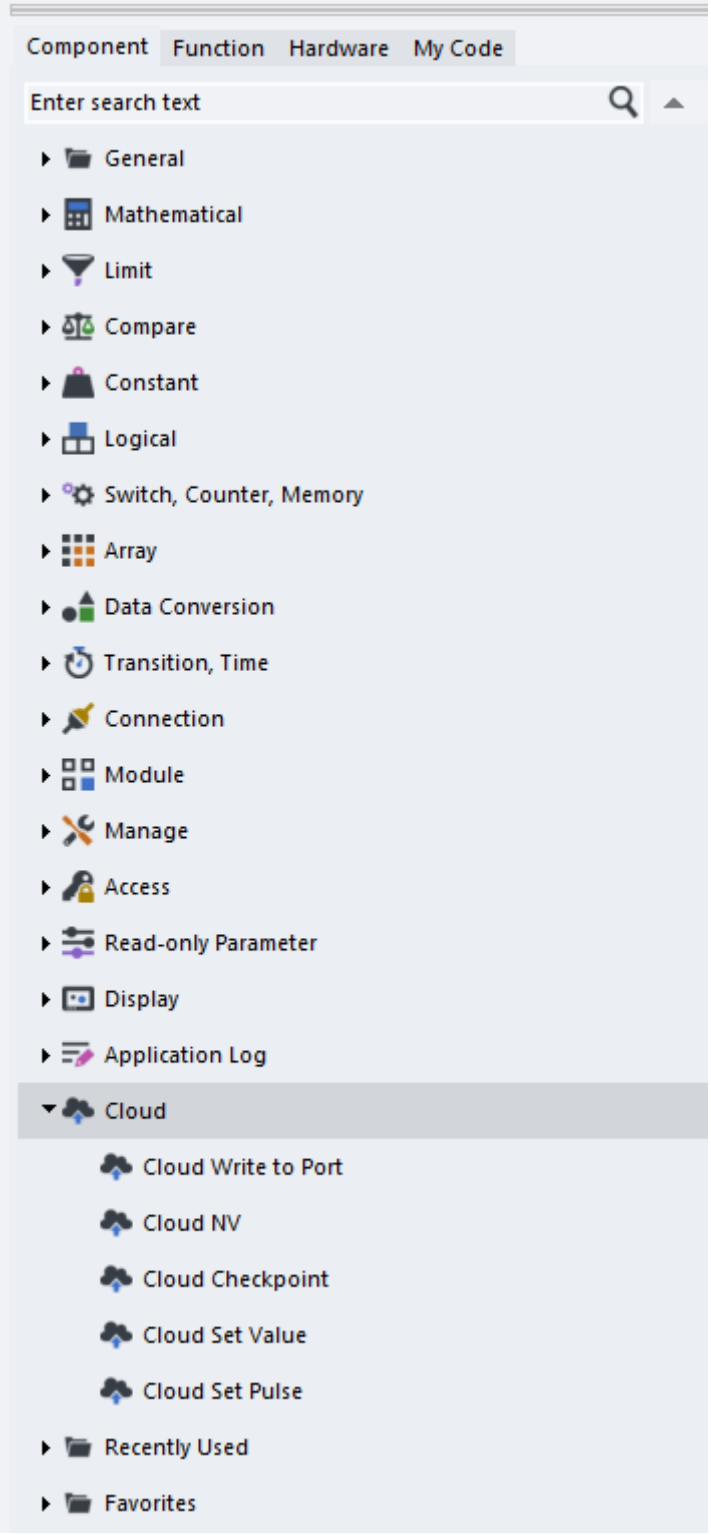
Define Application Log Page

Use: Implements application data logging; use with the **Define APPAreas** component.

The **File menu's** > **Import Page** > **Import Block** commands do not work with this component. Importing a **Define APPLog** page strips the page of its contents.

Programming

Cloud Menu



Programming

Cloud Write to Port

Use:

- Logs data points to the cloud
- Also works as a normal Checkpoint

Function:

- A1 = The data to be logged
- A2 = Data on A1 sampled when True (the data may be buffered until it can be sent to the cloud)
- A3 = The port, connect to the Cloud port defined by the hardware file—refer to your hardware's Application Interface (API) specification
- X1—if:
  - X1 = True, data sampling pending
  - X1 = False, no data sampling pending

Valid Connections

Pin/Property	Data Type/Comment
A1	EXTENDED
A2	BOOL
A3	PORT
X1	BOOL
V1 Signal	The NV name, must be unique in application (When combined with Namespace if applicable)
V2 Read Access	The Read Access level; '0'-'9' or '_'
V3 Write Access	The Write Access level; '0'-'9' or '_'
V4 Namespace Enabled	True or False ('T' or 'F')
V5 Cloud Enabled	True or False ('T' or 'F'). If False, this Component is basically a simple Checkpoint.

Cloud Checkpoint

Use:

- Creates a checkpoint signal that you can check with the PLUS+1® Service Tool program
- Route the signal that you want to check to A1; your **?** entry is the signal name
- Click the underscore (_) with the Query/Change tool to set a 0–9 PLUS+1 Service Tool program access level
- Namespace (NS) support can be toggled between True/False (T or F)
- Cloud support can be toggled between True/False (T or F)

The **?** names used in checkpoint and non-volatile dynamic memory components must be unique. Identical **?** names produce a compile error.

Function: **?** = User-defined name for the A1 signal, optionally with Cloud support.

Programming

Valid Connections

Pin/Property	Data Type/Comment
A1	EXTENDED
V1 Signal	The Checkpoint name, must be unique in application (When combined with Namespace if applicable)
V2 Read Access	The Read Access level; '0'-'9' or '_'
V3 Namespace Enabled	True or False ('T' or 'F')
V4 Cloud Enabled	True or False ('T' or 'F')

Cloud NV

Warning

All non-volatile memory has a limit to the total number of times that you can erase from and write to a memory sector. Non-volatile memory can become corrupted when you exceed this erase/write limit.

This corruption can change the values that your application reads from non-volatile memory. Changed values in your application can cause sudden, unexpected machine movements, equipment damage, and personal injury.

To reduce the risk of non-volatile memory corruption, you should:

- For more information, see *PLUS+1® Controller Technical Information*, BC152886484710, to find the lifetime erase/write rating of the non-volatile memory that is used in your controller.
- Develop your application so that no value written to non-volatile memory will exceed this erase/write rating during the lifetime of your application.
- Use a **Positive Transition** component with this memory component to prevent accidental continuous erase/write cycles.

Use:

- Read and write values to non-volatile (NV) memory
- On hardware power-up, the value in NV memory copies to X1
- Creates a parameter that you can read and write with the PLUS+1 Service Tool program
- **Namespace** (NS) support can be toggled between True/False (T or F)
- Cloud support can be toggled between True/False (T or F)

The **?** names used in checkpoint and non-volatile dynamic memory components must be unique. Identical **?** names produce a compile error.

Function:

- **?** = Your name for the NV memory value
  - When used in a page that has a **Namespace** value, the name of the NV memory value will be prefixed with the **Namespace** value if the **Namespace** (NS) property is set to True (T)
- If A1 is True and X2 is False, then the X1 value is written to NV memory
- If A2 is True, then the value in NV memory is copied to X1
- If A3 is True and A2 is False, then the A4 value is copied to X1, independent of X2 status

Programming

  - A3 has no effect if A2 is True

- A4 is the New value for X1 to use if A3 is True and A2 is False
- X1 is the Value read from NV memory/value to be written to NV memory
- X2 = True indicates that writing of X1 value to NV memory is pending completion

Service Tool interaction:

- If the read access level permits, the Service Tool can read from the NV memory using the Name set on this component (including **Namespace** if any)
  - The Service Tool will read the value from NV memory, **not X1**.

    The Service Tool will read the value from NV memory, **not X1**.

- If the write access level permits, the Service Tool can write to the NV memory using the Name set on this component (including **Namespace** if any)
  - **Warning**

    The value will not only be written to the NV memory, **X1 will also be modified in this case to match the new content in NV memory.** This modification of X1 happens in the very beginning of the loop, before the component is executed.

- In conclusion: reading and writing this NV memory parameter from the Service Tool is not handled in a symmetrical fashion, and you must be aware of this difference when designing the application.

Valid Connections

Pin/Property	Data Type/Comment
A1	BOOL
A2	BOOL
A3	BOOL
A4	INT, BOOL
X1	EXTENDED
X2	BOOL
V1 Signal	The NV name, must be unique in application (When combined with Namespace if applicable)
V2 Read Access	The Read Access level; '0'-'9' or '_'
V3 Write Access	The Write Access level; '0'-'9' or '_'
V4 Namespace Enabled	True or False ('T' or 'F')
V5 Cloud Enabled	True or False ('T' or 'F')

Cloud Set Value

Use:

- When experimenting with values
- Inputs values directly from the PLUS+1® Service Tool program to the controller without having to use memory components to read and write values
- Turning off the controller returns all values to 0
- Click the underscore (_) with the Query/Change tool to set a 0–9 PLUS+1 Service Tool program access level

Programming

Function: **?** = User name for the value; use this name to access the value in the PLUS+1 Service Tool program

Valid Connections

Pin/Property	Data Type/Comment
X1	EXTENDED
V1 Signal	The Set Value name, must be unique in application (When combined with Namespace if applicable)
V2 Write Access	The Write Access level; '0'-'9' or '_'
V3 Namespace Enabled	True or False ('T' or 'F')
V4 Cloud Enabled	True or False ('T' or 'F')

Cloud Set Pulse

Use:

- Applies a True/False pulse for one program loop when activated through the PLUS+1 Service Tool program

Function: **?** = User name for the pulse; link this name with the pulse pushbutton in the PLUS+1 Service Tool program

Valid Connections

Pin/Property	Data Type/Comment
X1	BOOL
V1 Signal	The Pulse name, must be unique in application (When combined with Namespace if applicable)
V2 Write Access	The Write Access level; '0'-'9' or '_'
V3 Namespace Enabled	True or False ('T' or 'F')
V4 Cloud Enabled	True or False ('T' or 'F')

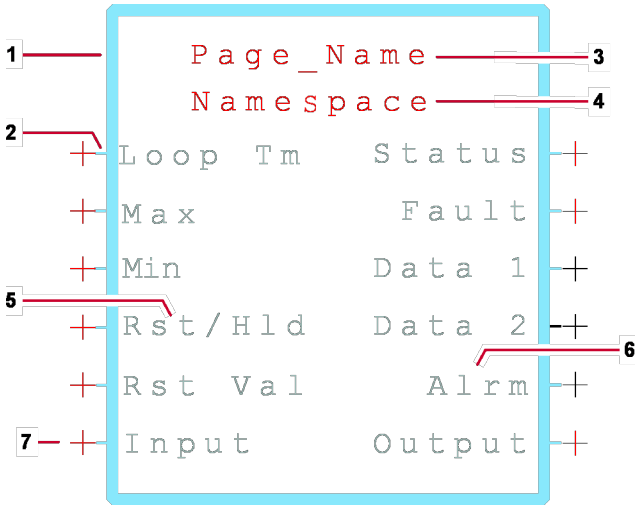
Programming

Page Layout Guidelines

This following contains guidelines for laying out pages and labeling the ports on the pages.

Leave the **Layer** value of the **@PAGENAME** placeholder set to the **Layer** value of **PageName**. Changing this value may cause the page name to disappear. It can also cause compile problems and problems with the PLUS+1® GUIDE program's advanced features.

The following figure and table have layout guidelines that you can use when creating the Page View in the Page Interface Editor.



Page View layout guidelines

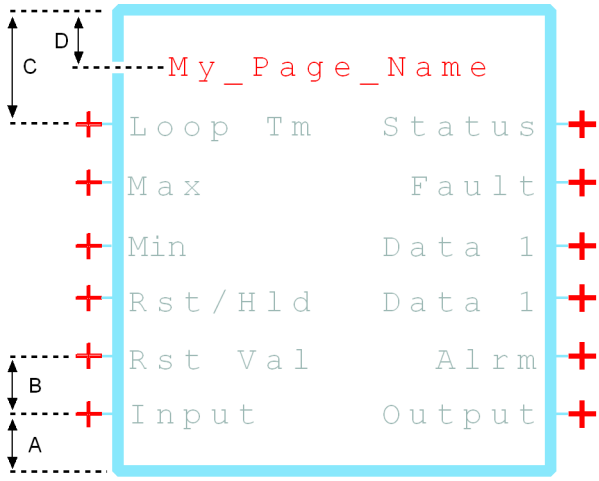
Item	Element	Layer/Color	Alignment	Other
1	Border	SymbolBodyThin/Blue	——	Width: 1.0 mm
2	Pin	SymbolBodyThin/Blue	——	Style: Solid
3	Page name	PageName/Red	Center on block	Title caps; underscore (_) spaces between words; do not overrun page borders
4	Namespace[1]	Namespace/Red	Center on block	Title caps; underscore (_) spaces between words; do not overrun page borders
5	Input label	CoverSheet3/Grey	X left, Y center	Title caps; do not underscore (_) spaces between words
6	Output label	CoverSheet3/Grey	X right, Y center	Title caps; do not underscore (_) spaces between words
7	Insertion point	——	——	Always lower left

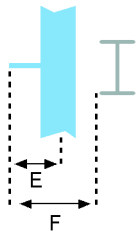
[1] The display of a NameSpace value is optional.

Text size guidelines

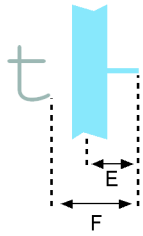
Item	Font	Cell Height	Cell Width	Stroke Width	Relative Size
Page name	Standard System	2.5000	1.6664	0.3750	100 %
Input label	Standard System	2.5000	1.6664	0.3750	100 %
Output label	Standard System	2.5000	1.6664	0.3750	100 %

Programming





Detail: Input label and connection



Detail: Output label and connection

Page View dimensions

Item	Dimension	Item	Dimension	Item	Dimension
A	5.0 mm	C	10.0 mm	E	2.5 mm
B	5.0 mm	D	3.75 mm	F	3.75 mm

Programming

Port Label Abbreviations

Port Label Abbreviations—2 wheel to Right

Label	Abbreviation	Label	Abbreviation	Label	Abbreviation
2 wheel	2Whl	Equal	Eq	Maximum	Max
4 wheel	4Whl	Error	Err	Middle, Mid	Mid
Accelerate	Acel	Fail	Fail	Minimum	Min
Alarm	Alrm	Fault	Flt	Negative	Neg
All	All	Feedback	Fdbk	Neutral	Neut
Angle	Ang	Forward	Fwd	Not	Not
Average	Avg	Found	Fnd	Number	Nmbr
Buzzer	Buzz	Four wheel	4Whl	Operator	Op
Bypass	Bpas	Frequency	Freq	Output	Out
Calibrate	Cal	Ground	Gnd	Pack	Pck
Clockwise	CW	Guard	Grd	Parameters	Para
Command	Cmd	Handle	Hndl	Pass	Pass
Control, Controller	Ctrl	High	Hi	Passive	Pasv
Coordinate	Cord	Hold	Hld	Point	Pt
Counterclockwise	CCW	Horn	Hrn	Positive	Pos
Deadband	Dbnd	Hysteresis	Hyst	Potentiometer	Pot
Decelerate	Dcel	Increase, Increment	Inc	Presence	Prs
Decrease, Decrement	Dec	Initialize	Init	Propel	Prpl
Default	Def	Input*	In	Pulse	Puls
Diameter	Dia	Left	L	Pulse pickup unit	PPU
Digital	Dig	Left front	LF	Pulse width modulation	PWM
Direction	Dir	Left rear	LR	Range	Rnge
Displacement	Disp	Length	Lgth	Read	Rd
Done	Done	Light Emitting Diode	LED	Reset	Rst
Down	Dn	Loop	Lp	Reverse	Rvs
Drive	Drv	Loop Time	LpTm	Revolution	Rev
Enable	Enbl	Low	Lo	Right	R

Port Label Abbreviations—Right Front to Write

Label	Abbreviation	Label	Abbreviation	Label	Abbreviation
Right front	Rf	Solenoid	Sol	Time	Tm
Right rear	Rr	Speed	Spd	Value	Val
Sample	Smpl	Start	Strt	Wheel	Whl
Select	Slct	Status	Stat	Width	Wdth
Sensor	Snsr	Steer, Steering	Str	Work	Wrk
Set	Set	Stop	Stop	Write	Wr
Setpoint	Stpt	Sweep	Swp		
Soft	Sft	Switch	Sw		

Programming

Port Label Unit Abbreviations

Unit Abbreviations

Item	Abbreviation	Item	Abbreviation
ampere	**A**	**miles per hour**	**mph**
centimeter	**cm**	**millimeter**	**mm**
connector	**c**	**minute (time)**	**min**
degree	**Deg**	**ohm**	**ohm**
degree Celsius	**C**	**pascal**	**Pa**
degree Fahrenheit	**F**	**pin**	**p**
foot	**ft**	**pounds per square inch**	**psi**
hertz	**Hz**	**radian**	**rad**
hour	**h**	**revolutions per minute**	**rpm**
inch	**in**	**second (time)**	**s**
kilogram	**kg**	**volt**	**V**
kilometers per hour	**kph**	**watt**	**W**
meter	**m**		

IEC61131-3 PLC Languages

PLUS+1® GUIDE supports the following PLC programming languages of the standard IEC61131-3:

- Structured Text (ST)
- Function block Diagram (FBD)
- Ladder Logic (LD)
- Sequential Function Chart (SFC)
- Instruction List (IL)

Instruction List (IL) is deprecated by IEC61131.

Program Organization Units (POUs) can be developed in GUIDE using any of those languages.

POUs developed with any of the standard PLC programming languages (ST, FBD, LD, SFC, IL) can also be imported to GUIDE using the standard PLCopenXML file format.

Additionally, and as an extension of the standard, GUIDE also supports developing POUs in C code. C code POUs are described more in depth in the chapter: *C Code in PLUS+1 GUIDE* on page 416

About PLC Data Types

PLUS+1® GUIDE IEC61131-3 Implementation supports all data types defined in IEC61131-3.

If a POU is called from PLUS+1® GUIDE code then the data type of input and output variables need to match the corresponding GUIDE Data type, defined in the *Data Types* on page 180.

PLUS+1® GUIDE Data type	IEC61131 Data type
BOOL	BOOL
U8	USINT, BYTE
S8	SINT
U16	UINT, WORD
S16	INT
U32	UDINT, DWORD
S32	DINT

Programming

PLUS+1® GUIDE Data type	IEC61131 Data type
U64	ULINT, LWORD
S64	LINT
F32	REAL
F64	LREAL
ARRAY[n]type	ARRAY[0..n-1] OF type
COLOR	COLOR[1]

[1] To have support for the COLOR data type in PLC code, it is necessary to add the file `GUIDE extensions.xml` to the project list of **PLCopenXML** files. This file can be added via a context menu option on the project node PLC Units.

Own Data Types

User defined struct, alias and enum types can be created.

About POUs

Program Organization Units (POUs) of type Function (F) or Function Block (FB) are supported. A POU can be developed using one of the IEC61131-3 defined programming languages Structured Text (ST), Function Block Diagram (FBD), Ladder Logic (LD), Sequential Function Chart (SFC) or Instruction List (IL). Alternatively, a POU can also be developed using C code, see *C Code in PLUS+1 GUIDE* on page 416.

POUs of type PROGRAM (P) are not supported to be executed as part of GUIDE compiled applications. They can however be imported in the form of PLCopenXML.

In order to be able to use such POUs, they must first be converted into FUNCTION_BLOCK type. This can be accomplished by using the context menu available by right-clicking on a PROGRAM POU and selecting the option “Convert to FUNCTION_BLOCK”.

Conversion of a PROGRAM POU into a FUNCTION_BLOCK POU will result in any VAR_GLOBAL and VAR_ACCESS variables in the POU being moved out into global variable lists in the containing PLCopenXML file.

Extension POUs

flip_endianness

Function flip_endianness(atleast_16bit_int) : atleast_16bit_int

The function flip_endianness takes an integer of at least 16 bits width and returns it with flipped endianness.

This function is an extension to the PLC standard.

Custom variable sections

The following custom variable sections can be used in the POU and global variable interface to declare variables.

VAR CHECKPOINT

If this section is used in the POU variable interface, then each variable will be added as a Checkpoint in the diagnostic data if the POU is called with the **Call Named POU** or **Show Named Screen** component.

If this section is used in the global variable interface, then each variable will be added as a global Checkpoint in the diagnostic data. This requires that at least one POU is called in the graphical code.

VAR SET_VALUE

If this section is used in the POU variable interface, then each variable will be added as a Set Value in the diagnostic data if the POU is called with the **Call Named POU** or **Show Named Screen** component.

Programming

If this section is used in the global variable interface, then each variable will be added as a global Set Value in the diagnostic data. This requires that at least one POU is called in the graphical code.

VAR SET_PULSE

If this section is used in the POU variable interface, then each variable will be added as a Set Pulse in the diagnostic data if the POU is called with the **Call Named POU** or **Show Named Screen** component.

If this section is used in the global variable interface, then each variable will be added as a global Set Pulse in the diagnostic data. This requires that at least one POU is called in the graphical code.

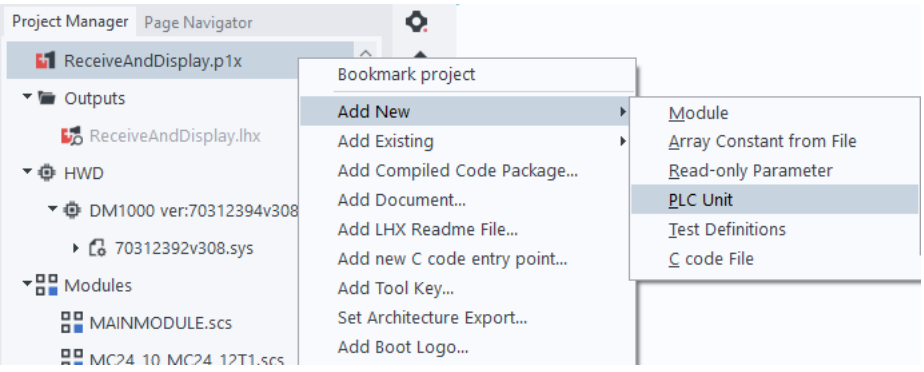
Create New PLC Unit and POU

POUs are stored in PLC Unit files following the IEC61131 standard file format PLCOpen XML. One PLC Unit can contain any number of POUs.

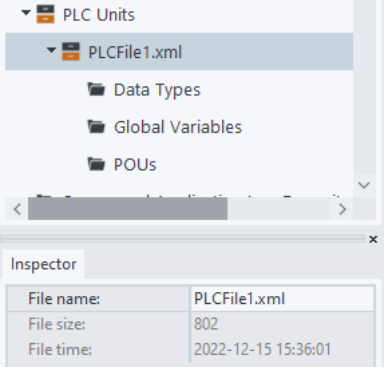
To add a new PLC Unit to the GUIDE project follow the steps below:

1. Right click on the Application in the project manager and select **Add New PLC Unit…** in the context menu:

   Manager > **Project Manager** > **Application** > **Add New PLC Unit…**



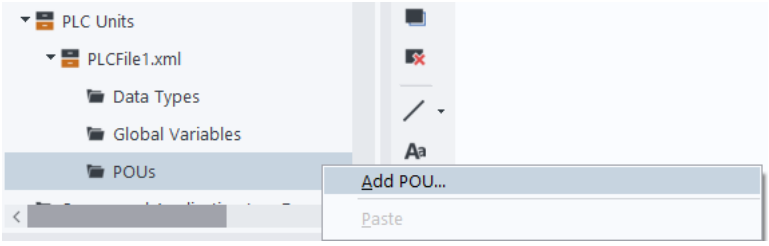
   A new folder **PLC Units** containing the new PLC Unit file will be added to the project manager.



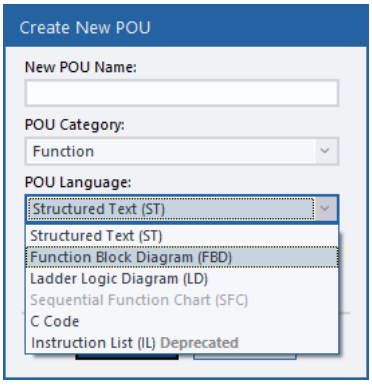
2. Optional: Edit the name of the PLC Unit file in the **Inspector**.
3. Right click on the POUs folder of the PLC Unit file in the project manager to add a new POU to a PLC Unit and select **Add POU...** .

   Manager > **Project Manager** > **PLC Units** > **POUs** > **Add POU…**

Programming



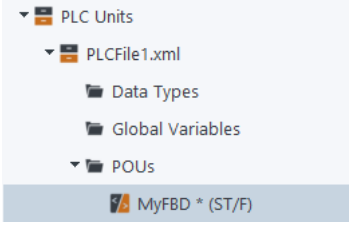
A “Create New POU” dialog will open. In the dialog the POU name can be edited.



The POU Category (Function or Function Block) and POU Language can be selected.

4. Double click the new POU in the Project manager to open the PLC Editor.

Manager > **Project Manager** > **PLC Units** > **POUs** > **the POU file**



Programming

Import Existing PLC Unit

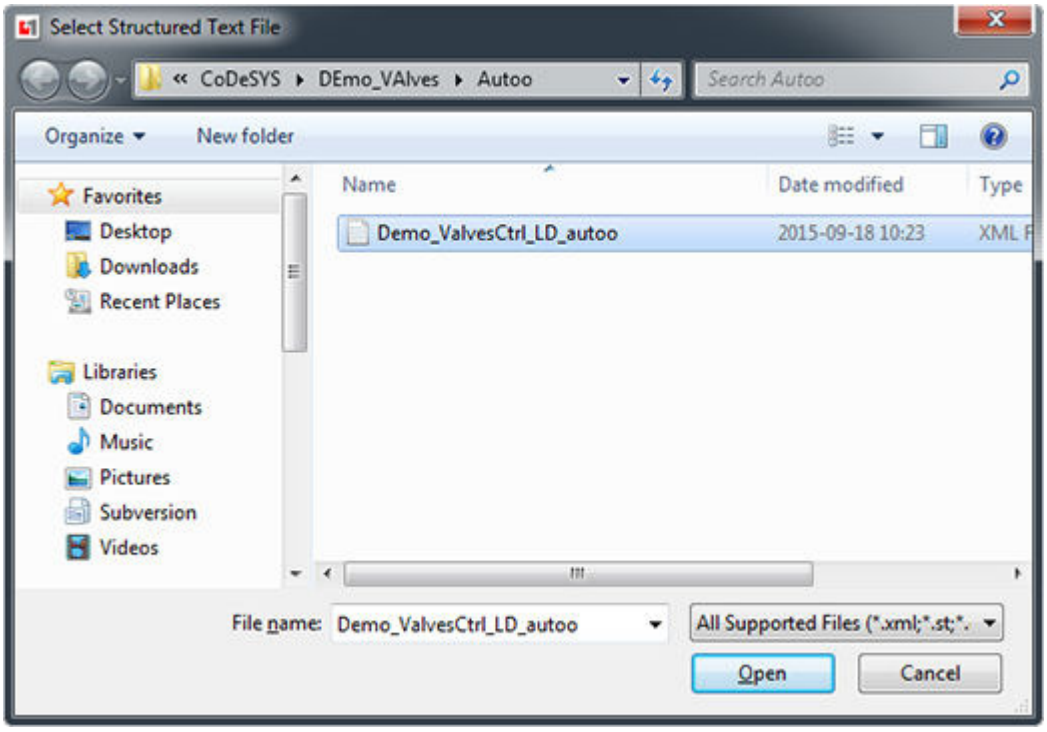
Any PLCOpen XML file with POUs compliant to IEC61131 can be imported into a GUIDE project.

Those files may come from a previously developed GUIDE project or any external tool that can export code to the PLCOpen XML format.

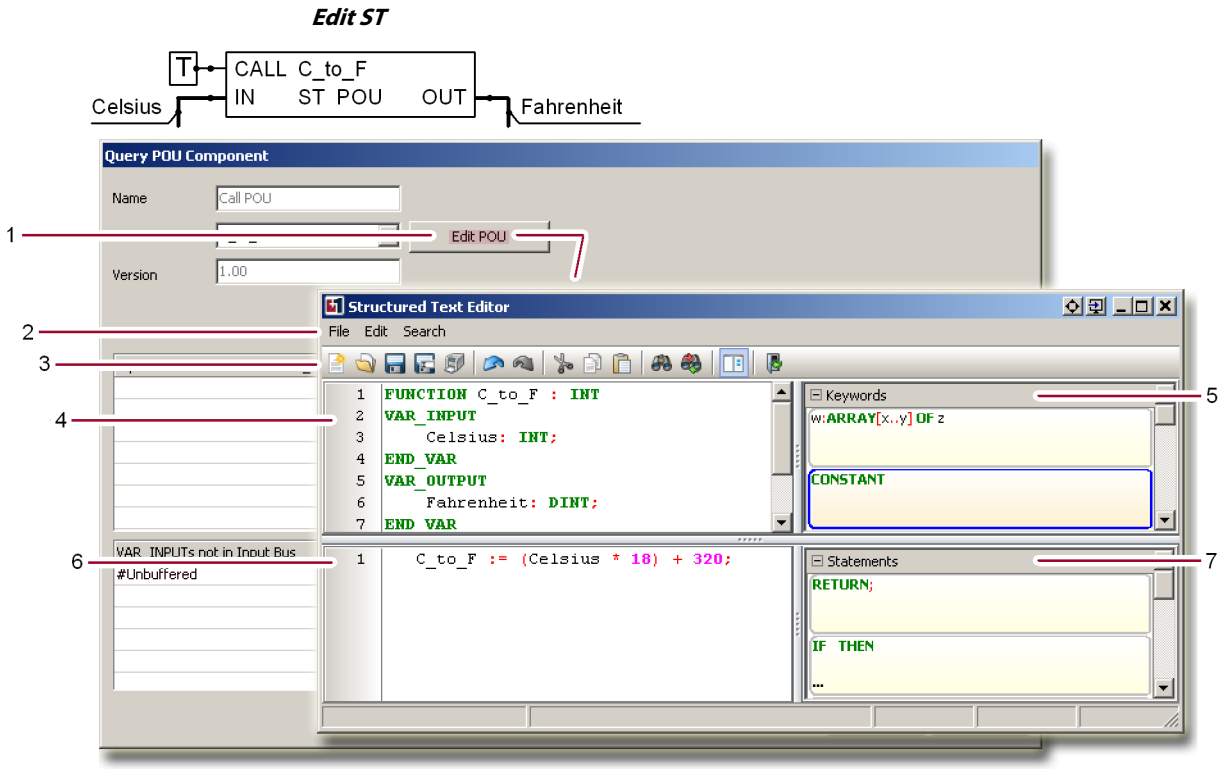
1. Right click on the Application in the Project Manager and select **Add Existing PLC Unit…** in the context menu:

   Manager > **Project Manager** > **Application** > **Add Existing PLC Unit…**

   A file dialog will be open.
2. Optional: Select the XML file to import and click **Open**.



Programming



The **Query POU Component** window in *Example 2* shows signals that are not in the **PLC Editor** window's **VAR_INPUT** and **VAR_OUT** declarations. In this example, these signals now populate the **PLC Editor** window's **VAR_INPUT** and **VAR_OUT** declarations.

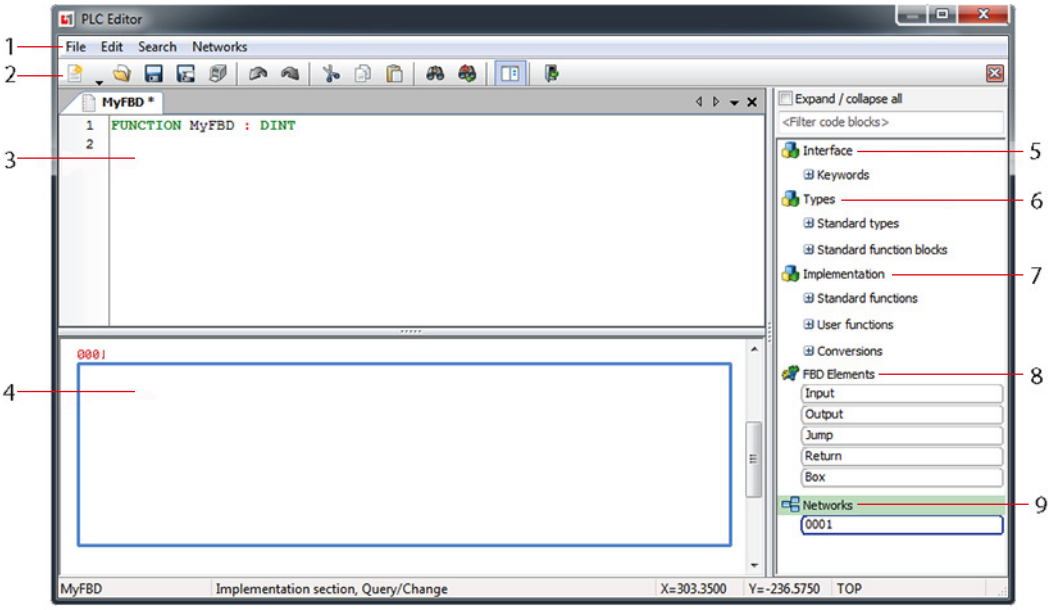
Example details

Item	Description
1	The **Edit POU** button opens the **PLC Editor** window.
2	Menu bar with standard file, edit and search commands.
3	Toolbar with standard file, edit and search buttons.
4	The Interface tab contains declarations. The **Query POU Component** window lists signals not already in the **PLC Editor** window's **VAR_INPUT** and **VAR_OUTPUT** declarations. Click the **Query POU Component** window's **Populate** buttons to add these signals to **VAR_INPUT** and **VAR_OUTPUT** declarations.
5	The Interface Code Blocks tab lists the declarations that can be dragged into the Interface tab.
6	The Implementation tab contains functions that return a value.
7	The Implementation Code Blocks tab lists the statements that can be dragged into the Implementation tab.

Programming

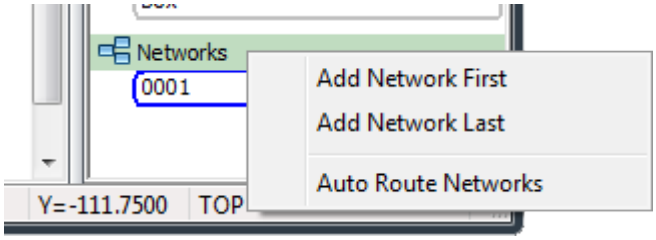
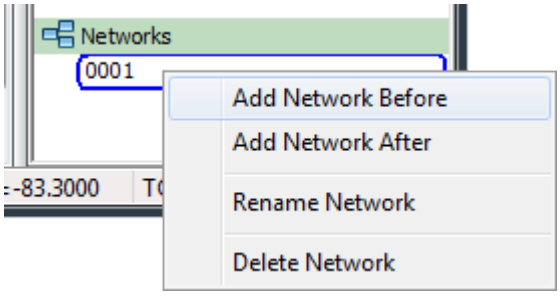
Edit FBD/LD

PLC Editor window



Item	Description
1	Menu contains **File**, **Edit**, **Search** and **Networks** management items.
2	Toolbar contains speed buttons to commonly used menu items.
3	Interface editor for editing the POU Interface, defining `VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT, VAR, CONSTANT`
4	Implementation section is the graphical GUIDE editor for FBD and LD. The TOP level contains one or several Network pages. Each Network contains FBD/LD code components connected in one network. The Label of the network can be modified by the user by using the Query function. To enter a network page use the Enter function (default 'E'). All editor commands that apply for GUIDE code are available; Like: `Enter(default 'E'),Query(default 'Q'), Move(default 'M'), Route(default 'R')` etc.
5	**Interface** in the Code Blocks section contains keywords that can be dragged to the Interface editor such as `VAR_INPUT..END_VAR`
6	**Types** in the Code Blocks section may be dragged to the interface editor or to the Implementation Section.
7	**Implementation** in the Code Blocks section contains all standard and user defined functions and function blocks that can be dragged to the implementation Section

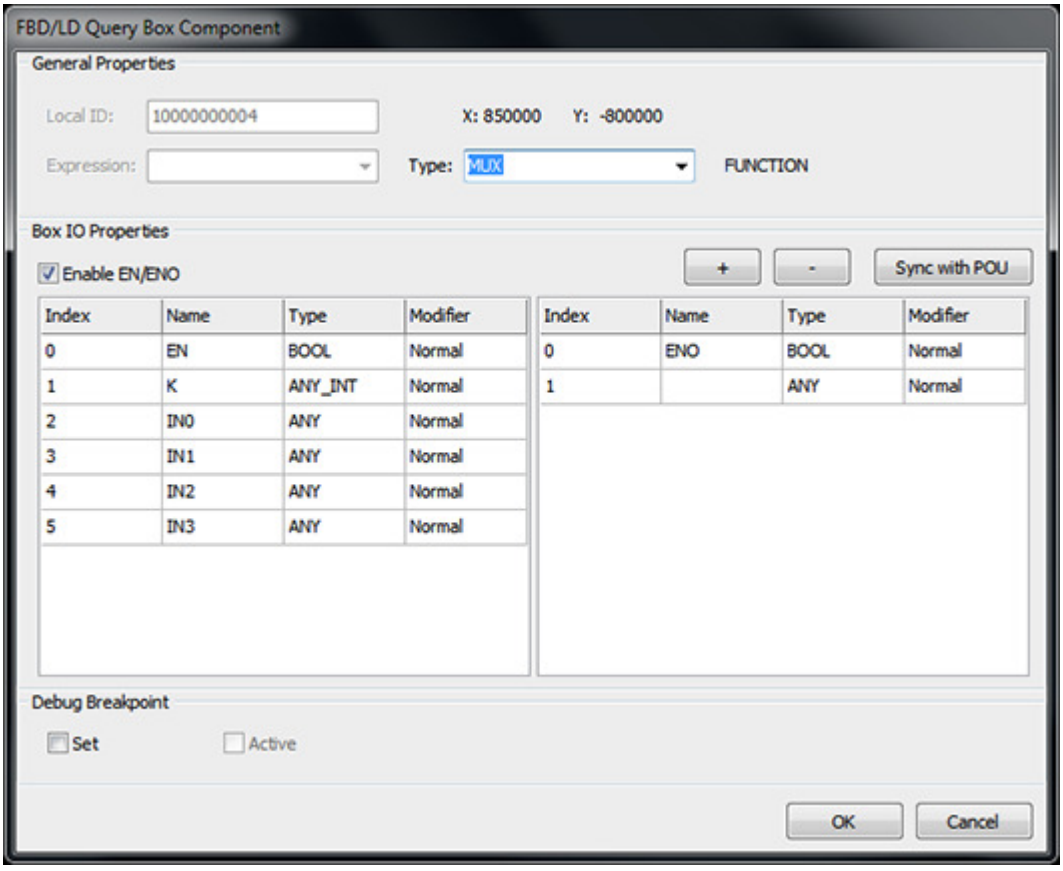
Programming

Item	Description
8	**FBD/LD Elements** in the Code Blocks section contains all available FBD and LD components that can be dragged into the available for FBD or LD that can be dragged into the Implementation Section.
9	The **Networks** navigator in the Code Blocks section navigate between different networks using the mouse double click. `Up` and `Down` arrow will navigate between **Networks**. `Right` and `Left` arrow will Enter and Leave the **Networks** Page. • Right Click on a Network in the **Networks** navigator will open a Context menu where Networks can be Added, or Auto Routed.  • Right Click on a Network in the **Networks** navigator will open a Context menu where Networks can be Added, Removed or Renamed. 

Programming

Querying Components

Querying an FBD/LD component will open the following dialog



The dialog above applies to "BOX" components. Simpler components will only have a subset of these options.

FBD/LD Query Box Component window description

Group	Name	Description
General properties	Local Id	Every component has a local id which is unique within the POU
	X, Y	Coordinates of the insertion point of the component
	Expression	The name of the variable associated with this component (if any). For function calls, this field will always be empty.
	Type	The data type associated with this component.
Box IO Properties	Enable EN/ENO	Switches between EN/ENO BOX and plain BOX component
	+	Add another input pin to the BOX component. Only applicable for extensible standard functions.
	-	Remove an input pin from the BOX component. Only applicable for extensible standard functions. (There is a minimum of 2 pins in most cases.)
	Sync with POU	Set the number of pins such that it matches the POU definition given by the Type field above.
Debug Breakpoint	Set	Set a breakpoint on this component. Only enabled when the user license allows debugging.
	Active	Can be used to enable/disable a breakpoint. Only enabled when Set is checked and enabled.

Programming

FBD/LD Networks

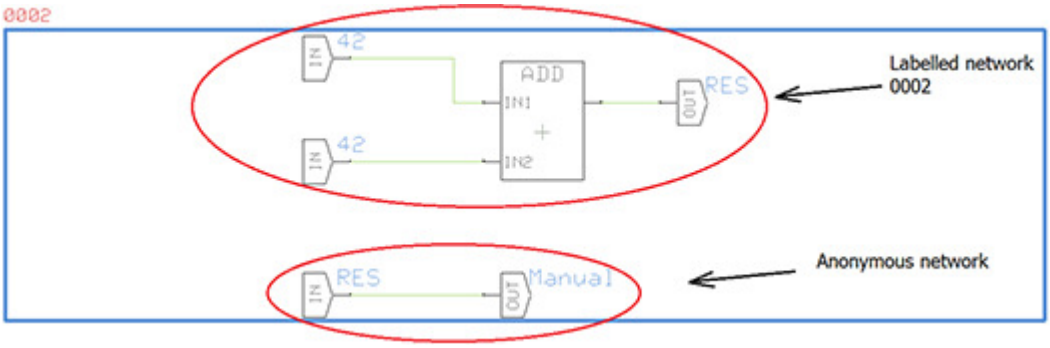
Each labelled network is a separate CAD page with a number of interconnected components and branches. All labelled networks can be used as a target for the Jump component.

Networks are evaluated as:

1. **signal flow**, i.e. from the output side of the component to the input side of the connected component (for FBD language)
2. **power flow**, i.e. analogous to the flow of the electric power, from left to right (for LD language)

If a page contains more than one set of interconnected components, each such set, starting from the second, is called an anonymous network. An anonymous network is an unlabeled network which is executed after a normal, labelled, network.

The following picture clarifies the definition:



If anonymous networks are present in a page, the usual left-to-right execution order cannot be applied straight ahead. Instead, networks will be executed one-by-one starting from the top to bottom and only within each network the left-to-right execution order is kept.

Caution

If a POU contains one or more anonymous networks, their execution order may be changed after running the Auto-router.

EN/ENO Components

Additional **Enable** (EN) input and **Enable Out** (ENO) output can be provided for Functions and Function Blocks in the IEC61131-3 languages.

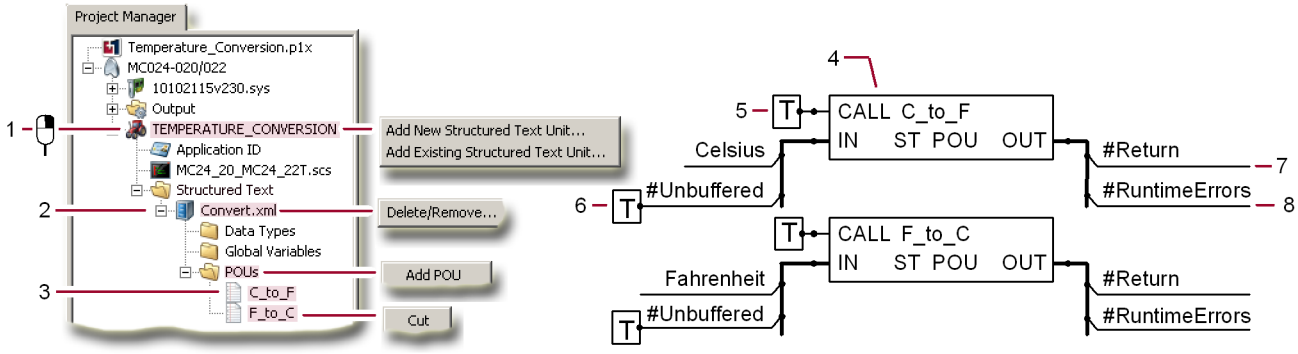
When these variables are used, the following rules apply:

EN is set to False	EN is set to True
• ENO is set to False. • Any operations of the function will not be executed. • Function outputs (VAR_OUTPUT, VAR_IN_OUT) will keep their values unchanged.	• ENO is set to True. • Operations defined by the function are executed. • Result of executed operations is assigned to outputs (VAR_OUTPUT, VAR_IN_OUT) of the function.

Programming

Use PLUS+1 GUIDE to Call Program Organizational Units (POUs)

The following example shows how POUs and PLCs are organized in PLUS+1® GUIDE, as well as how to call POUs from PLUS+1® GUIDE.



Example details

Item	Description
1	Right-clicks in the **Project Manager** tab open pop-ups that manage adding and removing PLC units from the project.
2	The file that contains the POUs (Program Organizational Units) that are used in the project. You can add existing PLC units or create new PLC units from within the PLUS+1® GUIDE software. Use the **Inspector** tab to change the default names of this file. You can add existing PLC units from these three file types: • ***.st** (plain text) • ***.exp** (plain text typically with multiple POUs). • ***.xml** (PLCopenXML standardized XML, typically with multiple POUs.) Any PLC units that you create within the PLUS+1® GUIDE software use the PLCopenXML format.
3	The individual POUs contained in the PLC file. Double-click a POU to open the **PLC Editor** window.
4	POU component in the PLUS+1 Drawing Area. A POU that is listed in the **Project Manager** tab that only uses basic types and arrays such as in/out variables can be called by one or several (or none) corresponding POU components in the Drawing Area, as well as from within another POU. Each component that calls a POU represents a unique instance of that POU.
5	When T, call the POU defined by this POU component.
6	An optional BOOL input that can set the POU to run unbuffered. This input only applies to array data. In most cases it is recommended to NOT connect this input unless you either don't use C code POUs or are certain that the C code POUs do not store pointers into GUIDE code data structures. • **#Unbuffered = T (constant TRUE)** The POU runs unbuffered. This setting is faster and uses less memory but comes with a risk that C code that is not written according to recommendations keeps pointers to arrays beyond their lifetime. However, if the called POU is not written in C, and does not call any POU written in C, then it is safe to set #Unbuffered to TRUE. • **#Unbuffered input = F (constant FALSE)** The POU runs buffered even in cases where the POU itself specifies that no buffer is needed. Running buffered is a safe choice but can use a bit more memory and execution time. • **#Unbuffered** input connected to something other than constant TRUE or FALSE (for example a BOOL variable, or even just an unconnected green wire). This is not how the **#Unbuffered** input should be used. The POU runs buffered except in cases where the POU itself specifies that it is safe to run unbuffered. • **No #Unbuffered input** This is the recommended use case. The POU itself will decide specifically which inputs/outputs will be buffered and which will not be buffered.

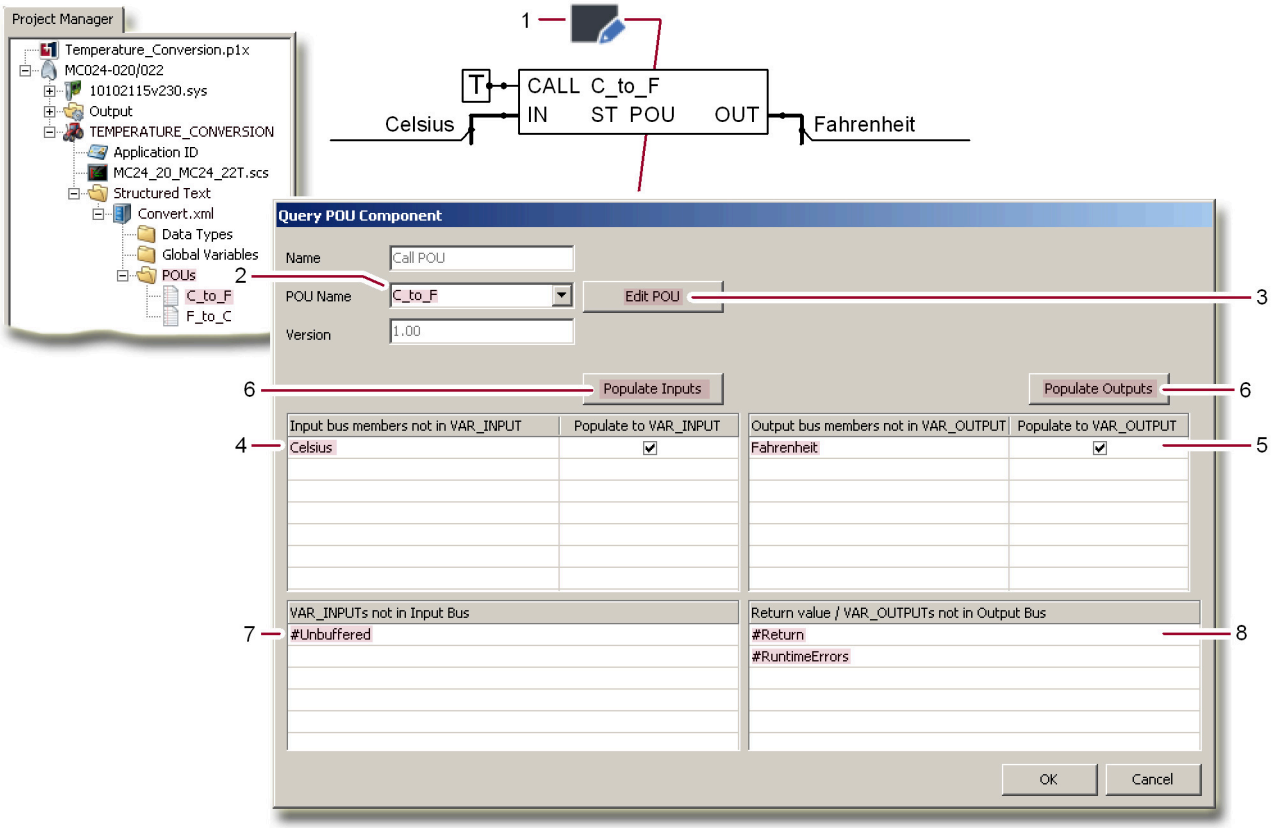
Programming

Example details (continued)

Item	Description
7	The default signal name for a return value from a POU with a **POU Type** of **Function**. You cannot give this signal another name.
8	A U16 output that can be used to report POU run-time errors. • 0x0000 = POU execution OK. • 0x0001 = Division by zero. • 0x0002 = Arithmetic overflow or underflow. • 0x0004 = Invalid subscript • 0x0008 = Assignment of a value that is outside the range of the data type. • 0x0010 = A “while”, “repeat-until”, backwards “jump” or variable-length “for” loop continued executing past its deadline and was prematurely stopped. This error does not apply to fixed-length for-loop • 0x0020 = Invalid argument (such as a negative value sent to the square root function). • 0x0040 = Null pointer error when trying to use a not-assigned pointer. (Automatically handled in runtime by using the type default value of the type pointed to.) To read out the current runtime error flags directly from PLC code, call the GET_CURRENT_RUNTIME_ERROR_FLAGS() function. This is a built-in PLC function, which returns a WORD.

Programming

Call from PLUS+1 GUIDE — Inside the POU Component



The signals applied to the **C_to_F** call **POU** component are changed here to better show features of the **Query POU** window.

Example details

Item	Description
1	Click the **POU** component with the Query/Change tool to open the **Query POU Component** window.
2	The **POU Name** drop-down lists the POUs that are shown in the **Navigation** tab. Select the POU that you want to call from this list.
3	The **Edit POU** button opens the **PLC Editor** window. While you can open and edit a POU through the Query tool and the **Edit POU** button (item **3**), double-clicking a POU in the **Project Manager** tab is the preferred method for opening a POU for editing.
4	Lists signals connected to the **POU** component's **IN** bus but not in the **PLC Editor** window's **VAR_INPUT** declarations.
5	Lists signals connected to the **POU** component's **OUT** bus but not in the **PLC Editor** window's **VAR_OUTPUT** declarations.
6	The **Populate Input**and **Populate Output** buttons add checked items (**Celsius** and **Fahrenheit** in this example) to the **PLC Editor** window's **VAR_INPUT** and **VAR_OUTPUT** declarations.
7	Lists the signals that are not connected to the POU component's **IN** bus but are listed in the **PLC Editor** window's **VAR_IN** declarations.
8	Lists the signals that are not connected to the POU component's **IN** bus but are listed in the **PLC Editor** window's **VAR_OUTPUT** declarations.

SFC POUs

SFC POUs are always Function Blocks since they have internal stored state. SFC POUs consists of several steps with associated actions. Add a new Action by right-clicking on either the Main SFC node, or the Actions node, in the Code Blocks tree and then select the **Add Action…** menu item.

Actions are controlled by Action Qualifiers described in the table below:

Programming

Action Qualifiers

Action Qualifier	Meaning	Used in combination with Duration
N	**N**on-stored This is the default Qualifier; If no Qualifier is specified for an Action, then N is assumed. Executed when a Step is Active. Also executed during Final Scan.	No
R	overriding **R**eset The Action is deactivated. This Qualifier can be used after "stored" Qualifiers (see "S", "SD", "DS" and "SL" for information) to deactivate Action execution.	No
S	**S**et (**S**tored) The Action is set to be executed at every cycle, starting with the cycle where the Step with the Qualifier becomes active, until it is deactivated by the "R" Qualifier. The Action is also executed on the cycle where "R" Qualifier deactivates the Action.	No
P	**P**ulse The Action is executed when the associated Step becomes active and then is executed again in the subsequent cycle, regardless of a transition. If the same Action is used in consecutive Steps, the Action will still execute twice only.	No
P1	**P**ulse (rising edge) Executes once when there is a rising edge detected for the Action Control block. The Action Control block does not always have a rising edge when the associated Step becomes active. For example, if two consecutive Steps use the same Action, then transitioning from the first Step to the second Step does not create a rising edge.	No
P0	**P**ulse (falling edge) Executes once when a falling edge is detected for the Action Control block. Similar to the "P1" Qualifier, the change in Step state, i.e. deactivation of the Step in this case, does not necessarily mean there is a falling edge. Using "P1" and "P0" Qualifiers in the same Step with the same Action is not equivalent to using "P" Qualifier. The difference is that an Action with "P0" Qualifier does not get executed when no transition happens while Action with "P" Qualifier is always executed in the subsequent cycle.	No
L	Time **L**imited When the Step becomes active the Action is set to be executed every cycle starting with the current cycle given that the Action remains active, and the set time limit has not passed. The Action then is executed again one last time during Final Scan.	Yes
D	Time **D**elayed The timer starts once the Step is activated. After the specified time has elapsed, the Action will be executed once, and then again in the next cycle during Final Scan. If the Step is deactivated before the specified time has passed, the Action will not be executed, neither in the current cycle nor in the next cycle during the Final Scan.	Yes
SD	**S**tored and time **D**elayed The timer starts when the Step is activated and once the user defined time has passed, the Action starts to be executed in every cycle even if the Step associated with the Action is no longer active. The execution of the Action can be stopped with the "R" Qualifier. The Action is executed one last time before "R" Qualifier deactivates the Action, and it no longer continues being executed.	Yes
DS	**D**elayed and **S**tored It works the same as "SD" Qualifier, the only difference being that Action will not start being executed if the Step associated with the Action is not active when the user specified time has elapsed.	Yes
SL	**S**tored and time **L**imited When the Step becomes active the Action is set to be executed every cycle starting with the current cycle as long as the set time limit has not passed, even if the Step is no longer active. The execution of the Action can be stopped with the "R" Qualifier. The Action is executed one last time before "R" Qualifier deactivates the Action, and it no longer continues being executed.	Yes

Programming

Action Qualifiers (continued)

Action Qualifier	Meaning	Used in combination with Duration
A	**Active** **Executed when a Step is Active. (It is identical to N, except it is not executed during Final Scan.)** This is a non-standard Qualifier.	**No**
E	**Entry** **The Action is executed immediately after the transition condition to the Step is evaluated to be TRUE.** **Unlike "P1" Qualifier, this Qualifier does not look for a rising edge for the Action Control block and instead executes the Action when the Step itself becomes active.** This is a non-standard Qualifier.	**No**
X	**Exit** **The Action is executed immediately after the transition condition to the next Step is evaluated to be TRUE.** **Unlike "P0" Qualifier, this Qualifier does not look for a falling edge for the Action Control block and instead executes the Action when the Step itself becomes inactive.** This is a non-standard Qualifier.	**No**

As a rule, there must be exactly one Transition placed between each Step in an SFC POU.

Add a new Transition by right-clicking on either the Main SFC node, or the Transitions node, in the Code Blocks tree and then select the **Add Transition…** menu item.

Both Actions and Transitions have access to all the variables of their parent SFC POU. Additionally, a Transition is defined as a Function which returns a Boolean value, whereas Actions are Function Blocks.

Definitions

Final Scan

The Final Scan means that an Action is executed one more time after the transition condition becomes TRUE. The Final Scan only happens if the associated Action Control block detects a falling edge. For example, if the same Action is used in two consecutive Steps, there will not be a Final Scan after the first Step since the Action Control block will not produce falling edge on its output even though the Step active state has changed.

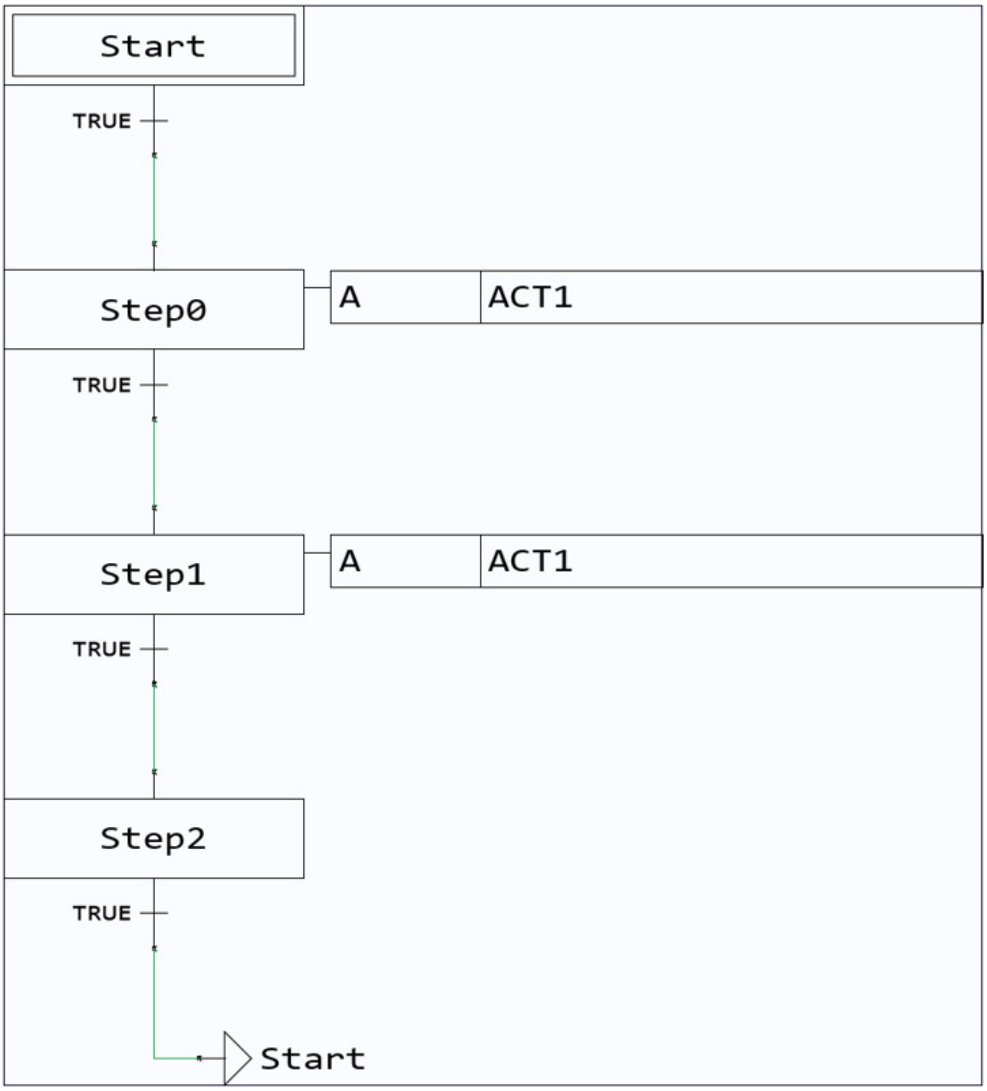
Examples

Examples below show execution of Actions based on the Qualifiers in each cycle (Loop Index).

"A" Qualifier

Programming

"A" Qualifier

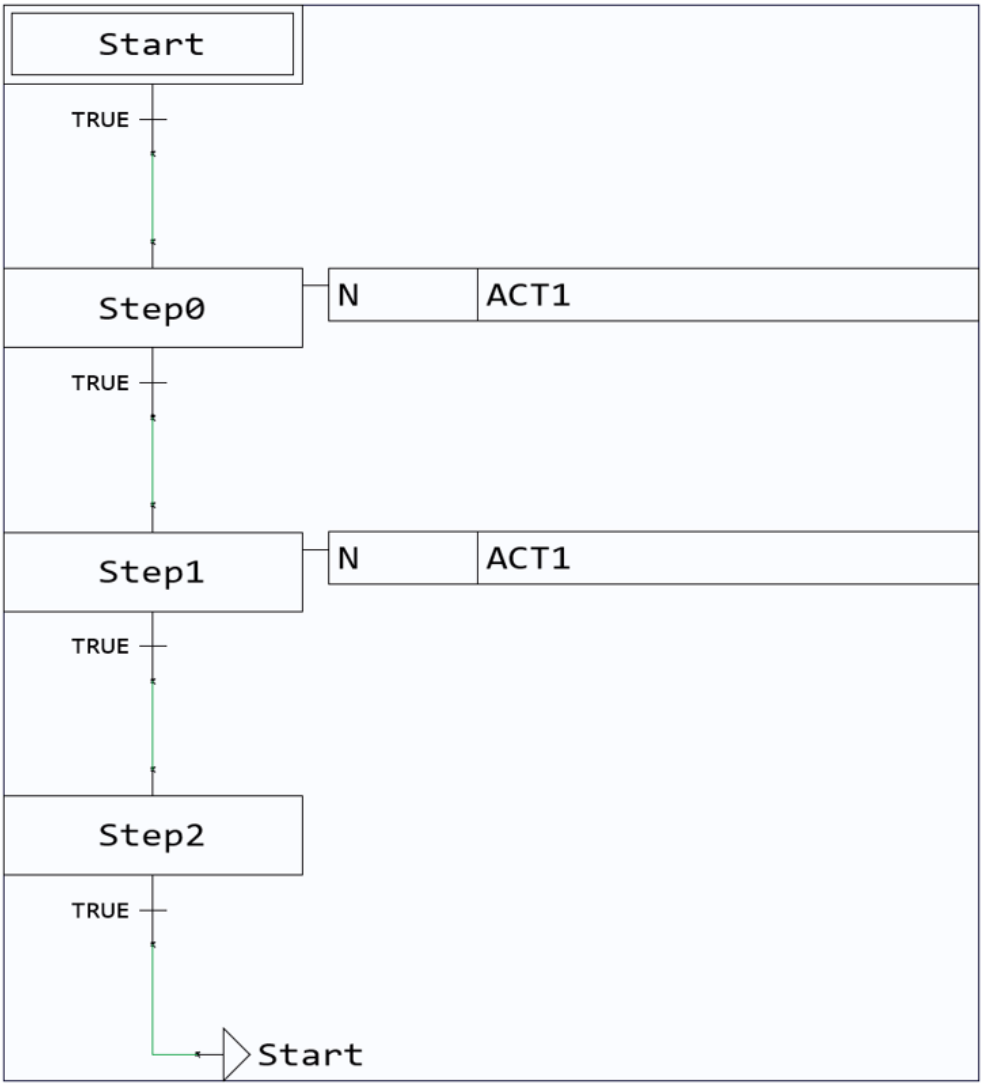


Loop Index	Start	ACT1
0	Start	Not executed
1	Step0	Executed
2	Step1	Executed
3	Step2	Not executed

"N" Qualifier

Programming

"N" Qualifier, example 1

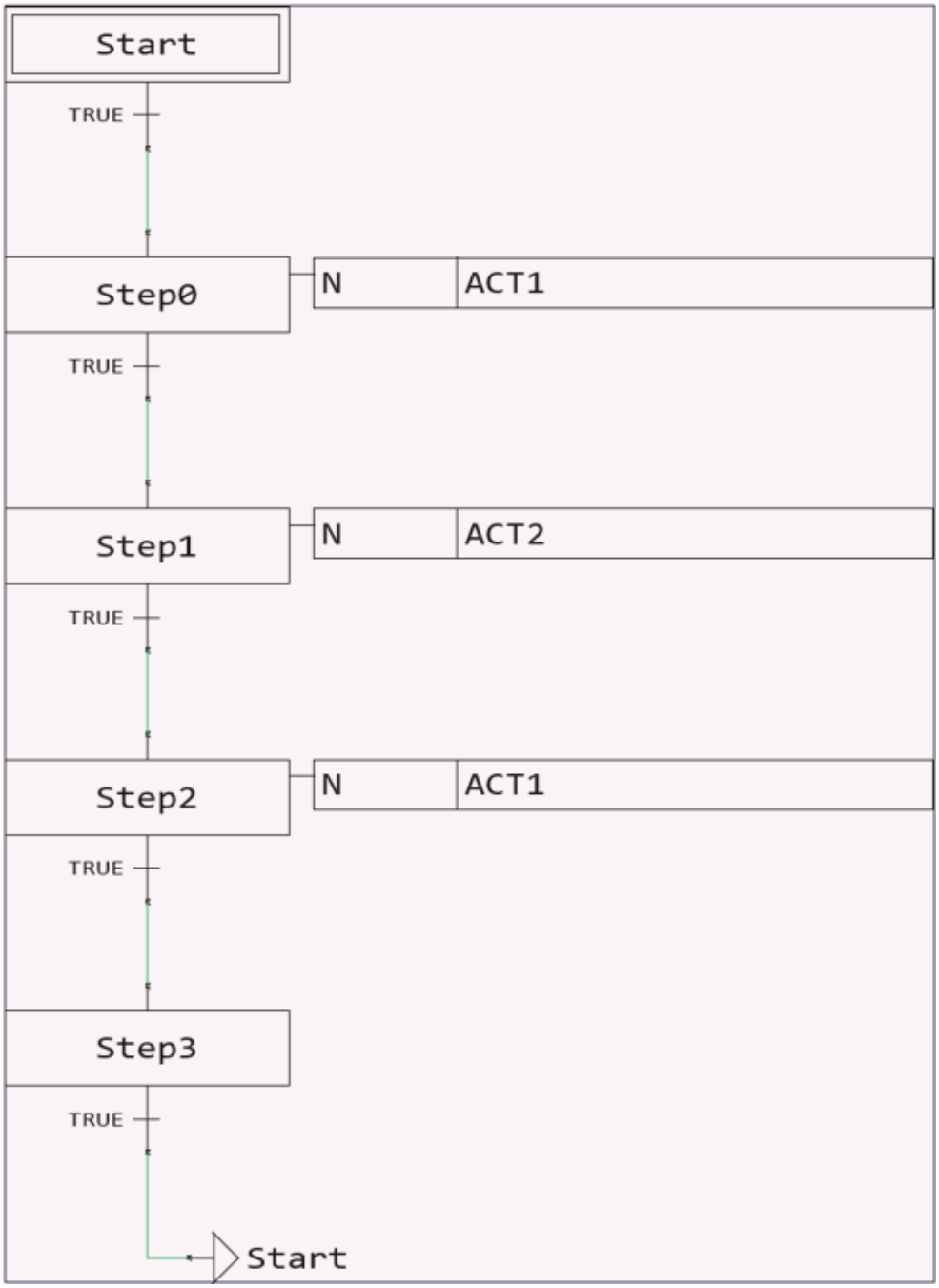


Loop Index	Start	ACT1
0	Start	Not executed
1	Step0	Executed
2	Step1	Executed
3	Step2	Executed *(Final Scan)*

In cycle 2, the Action is not executed once more in Final Scan since both "Step0" and "Step1" had the same Action and therefore no falling edge detected in the transition for the Action Control block

Programming

"N" Qualifier, example 2

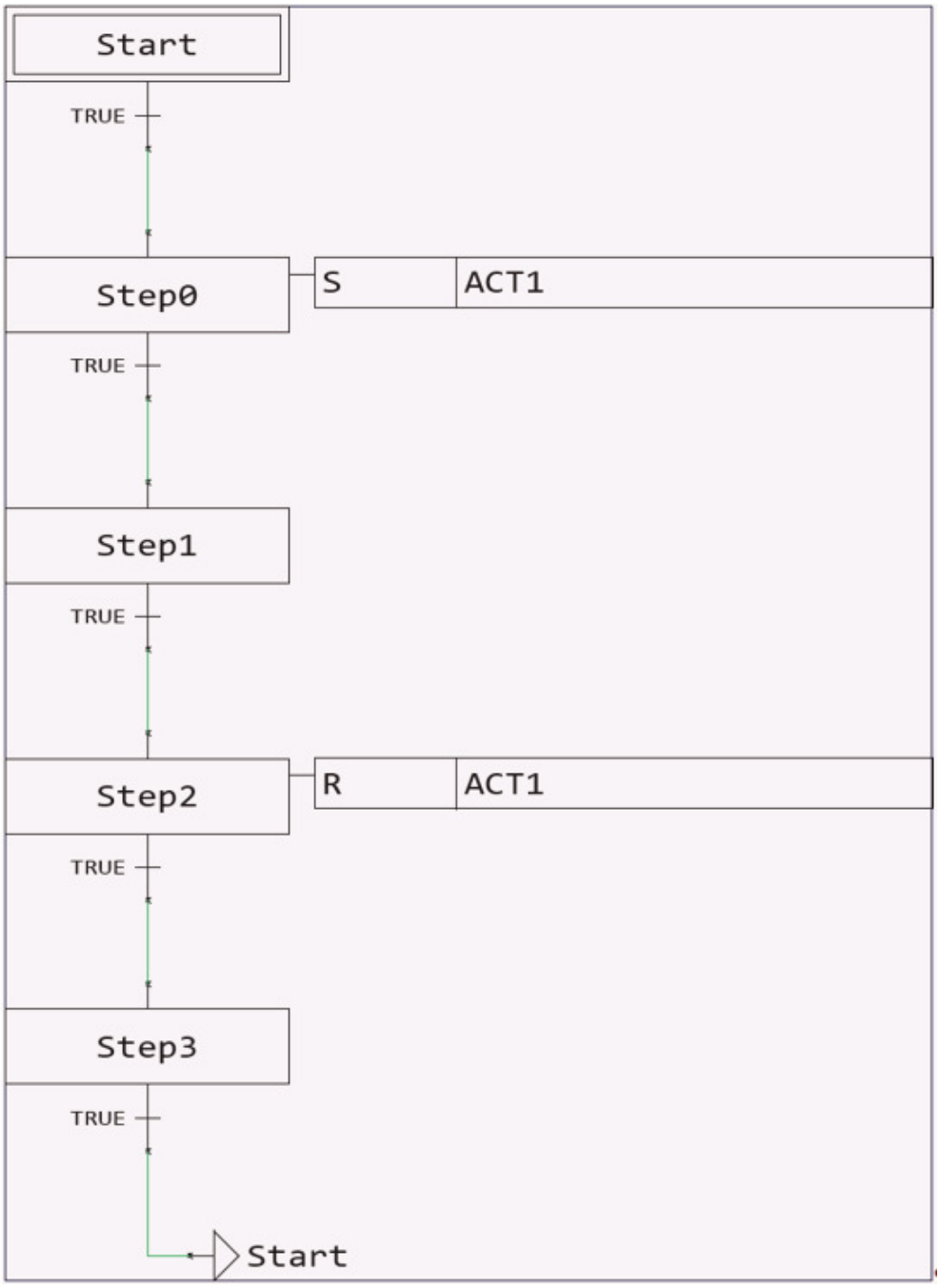


Loop Index	Start	ACT1	ACT2
0	Start	Not executed	Not executed
1	Step0	Executed	Not executed
2	Step1	Executed *(Final Scan)*	Executed
3	Step2	Executed	Executed *(Final Scan)*
4	Step3	Executed *(Final Scan)*	Not executed

Programming

"S" and "R" Qualifiers

"S" and "R" Qualifiers



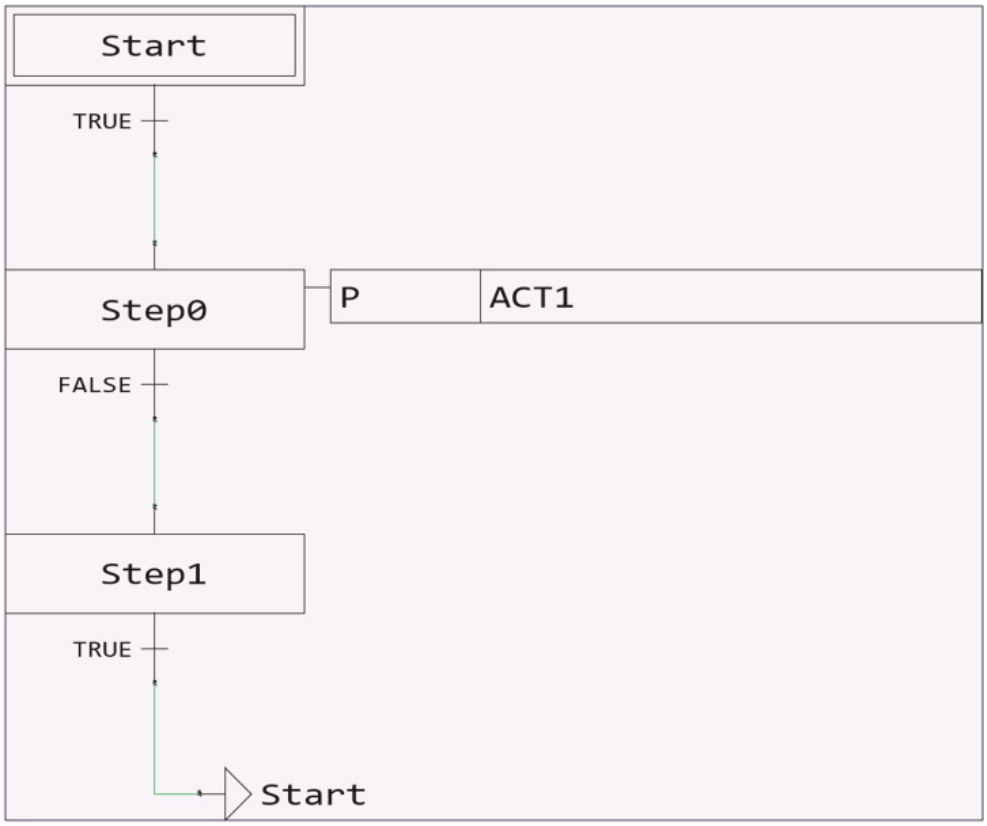
Loop Index	Start	ACT1
0	Start	Not executed
1	Step0	Executed
2	Step1	Executed
3	Step2	Executed
4	Step3	Not executed

Programming

Notice that the Action is also executed in cycle 3 where the Step with "R" Qualifier is activated.

"P" Qualifier

"P" Qualifier example 1

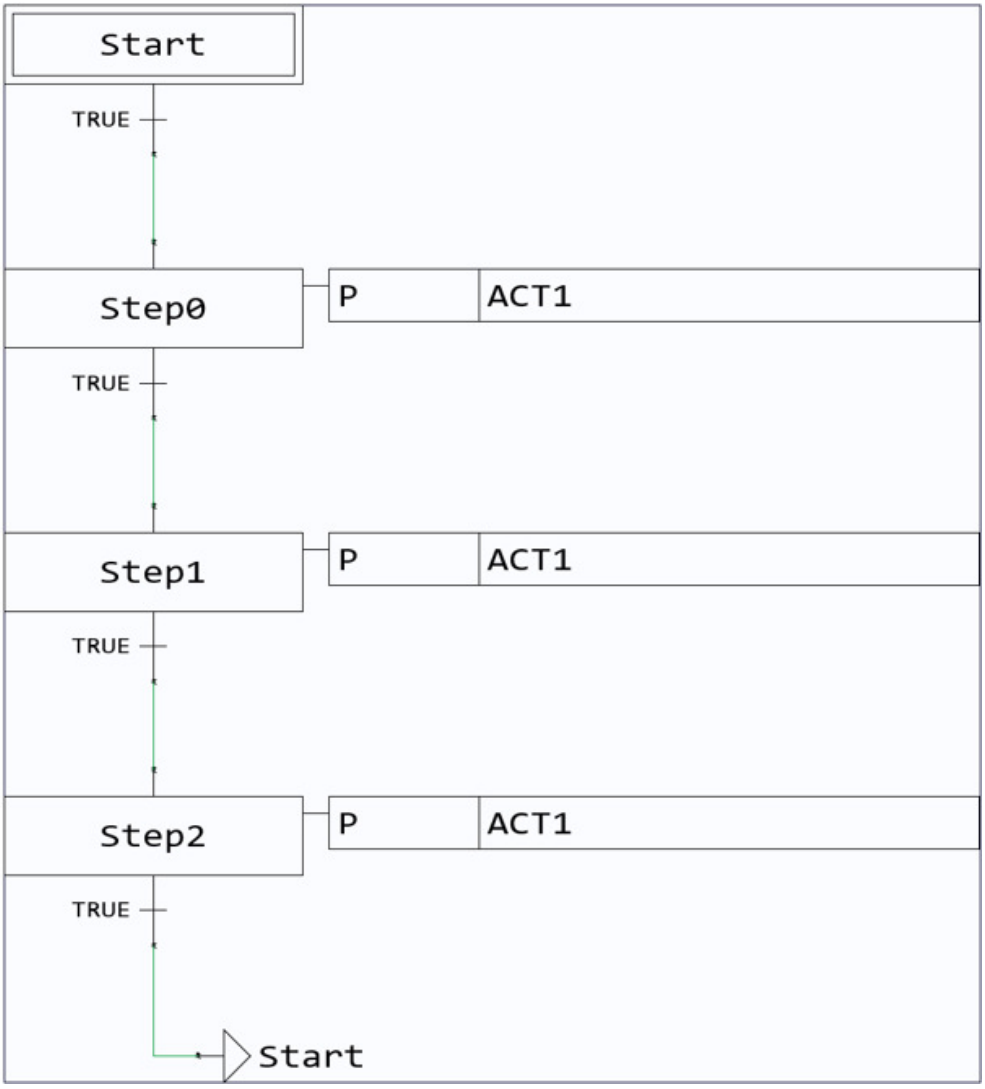


Loop Index	Start	ACT1
0	Start	Not executed
1	Step0	Executed
2	Step0	Executed
3	Step0	Not executed
4	Step0	Not executed
5	Step0	Not executed

Notice that in cycle 2 the Action is executed even though there was no transition.

Programming

"P" Qualifier example 2

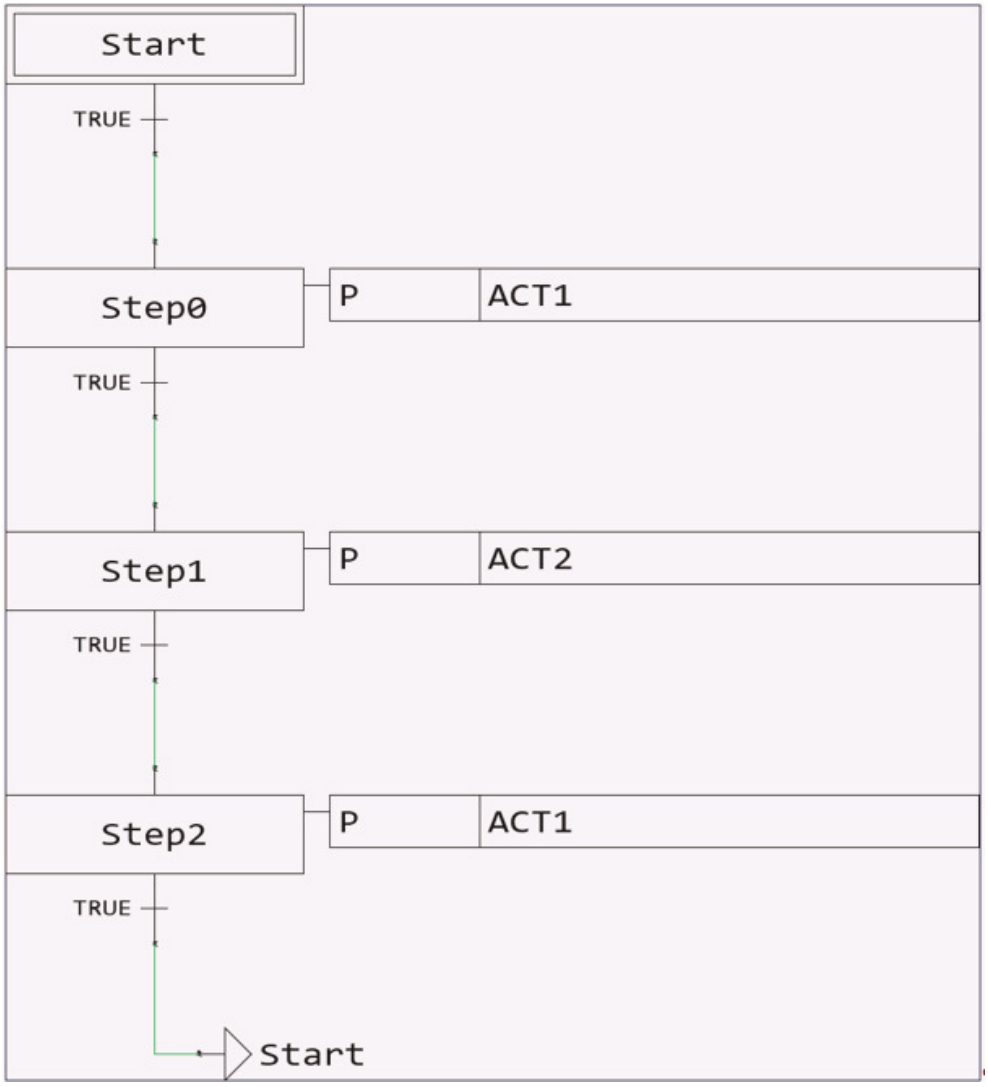


Loop Index	Start	ACT1
0	Start	Not executed
1	Step0	Executed
2	Step1	Executed
3	Step2	Not executed
4	Start	Not executed
5	Step0	Executed

Although there are 3 consecutive Steps with the same Action, the Action is executed only twice in a row.

Programming

"P" Qualifier example 3



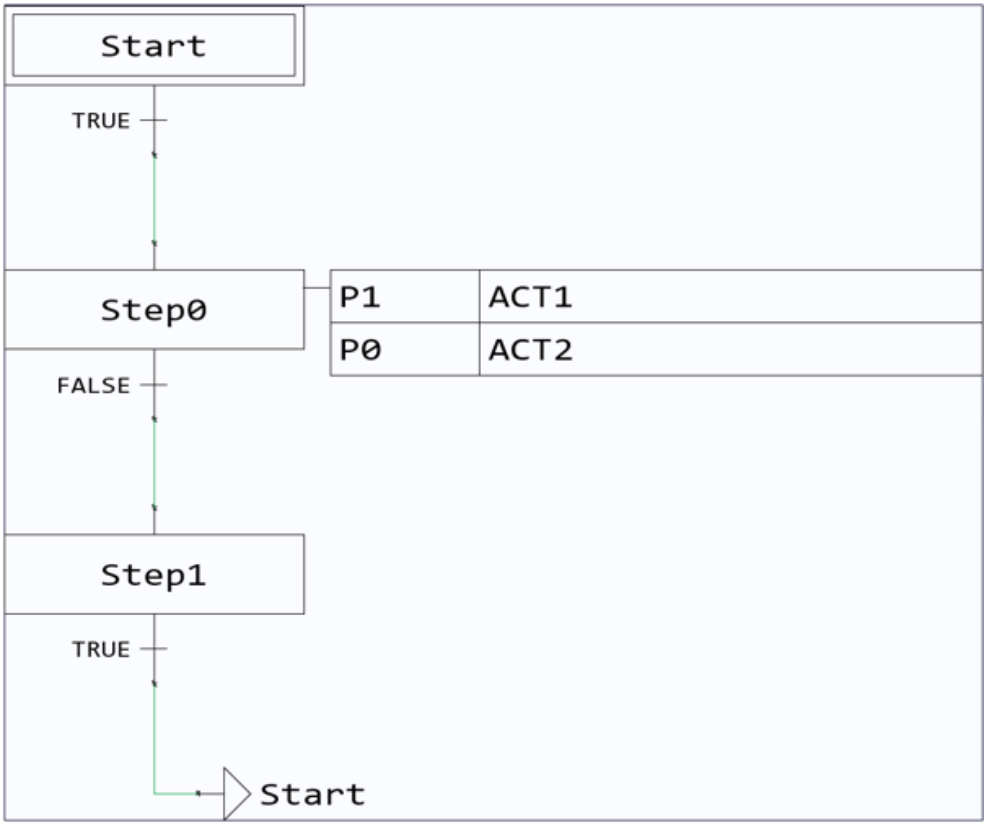
Loop Index	Start	ACT1	ACT2
0	Start	Not executed	Not executed
1	Step0	Executed	Not executed
2	Step1	Executed	Executed
3	Step2	Executed	Executed
4	Start	Executed	Not executed
5	Step0	Executed	Not executed

Unlike the previous example, in this case there is a different Action in "Step1", therefore "ACT1" is executed twice for each Step.

"P1" and "P0" Qualifiers

Programming

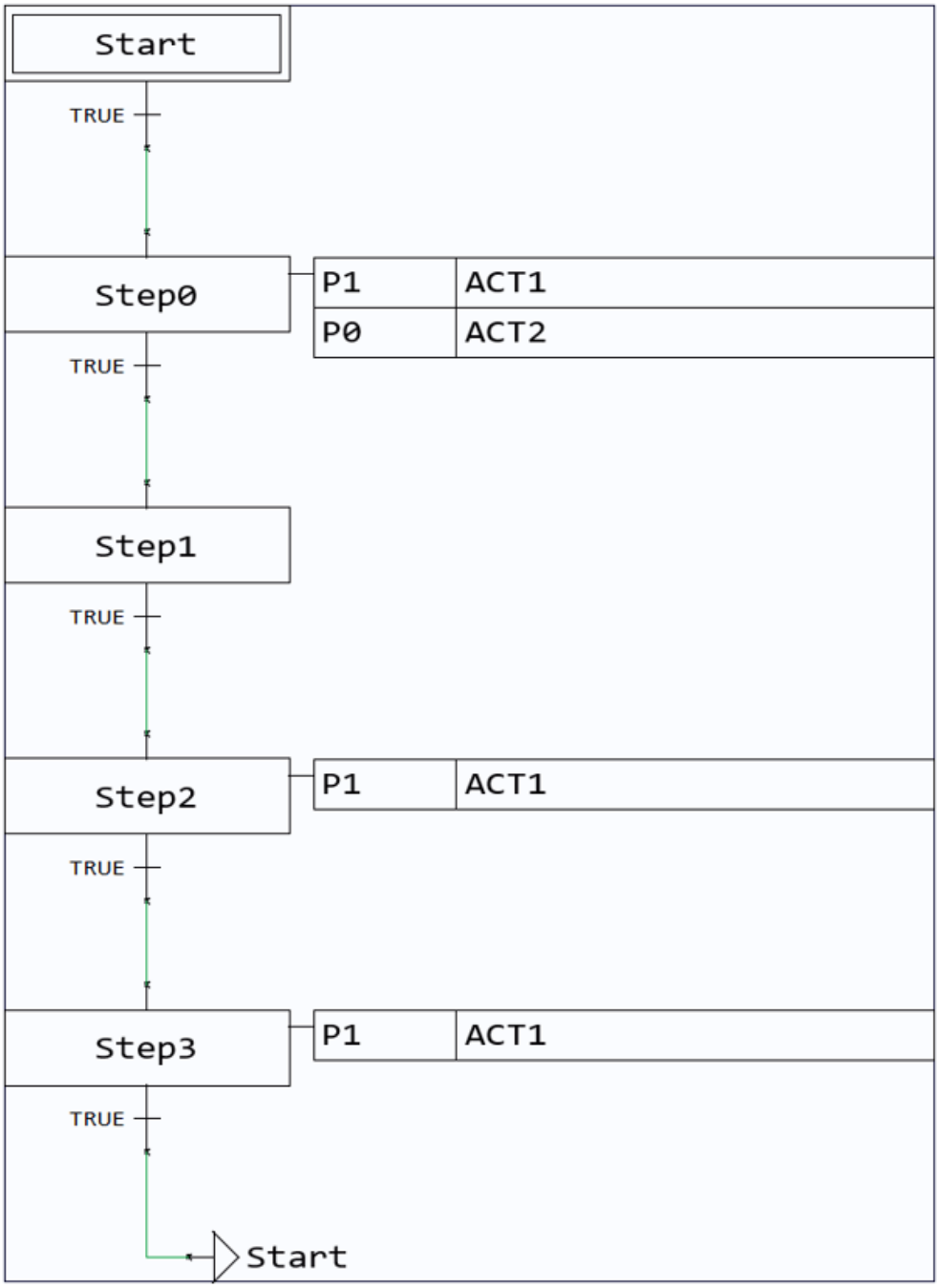
"P1" and "P0" Qualifiers example 1



Loop Index	Start	ACT1	ACT2
0	Start	Not executed	Not executed
1	Step0	Executed	Not executed
2	Step0	Not executed	Not executed
3	Step0	Not executed	Not executed

Programming

"P1" and "P0" Qualifiers example 2



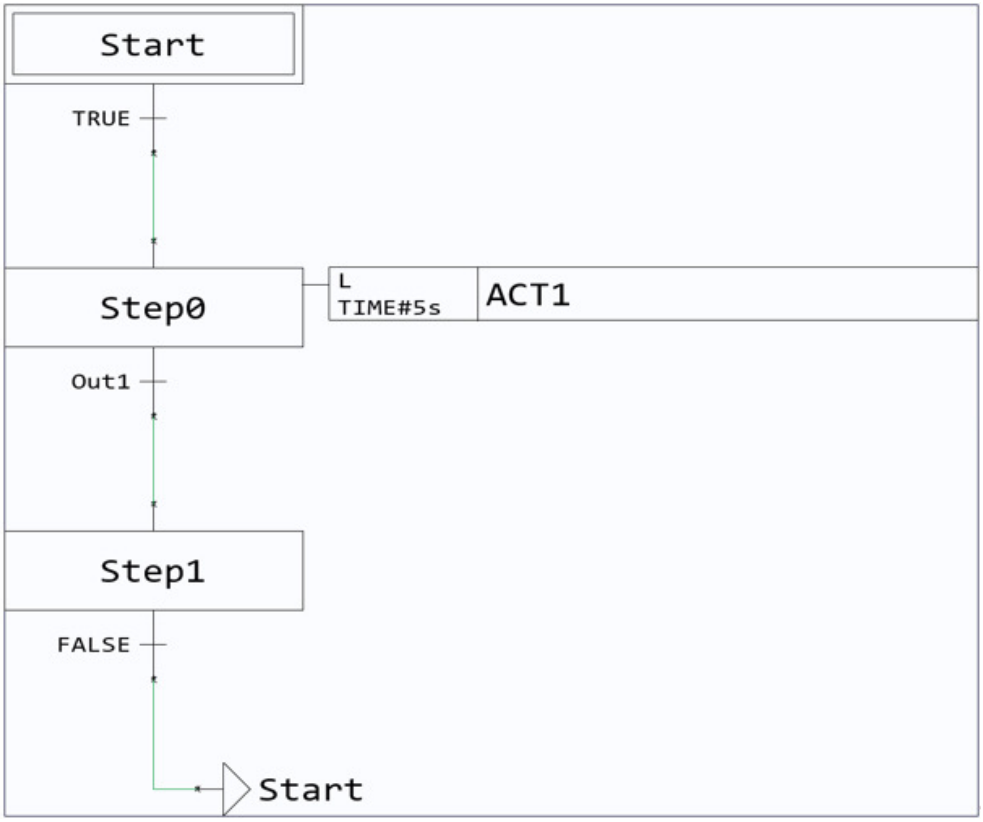
Loop Index	Start	ACT1	ACT2
0	Start	Not executed	Not executed
1	Step0	Executed	Not executed
2	Step1	Not executed	Executed
3	Step2	Executed	Not executed
4	Step3	Not executed	Not executed
5	Start	Not executed	Not executed

Programming

In cycle 1 there is a rising edge for "ACT1" Action Block, followed by a falling edge and rising edge in cycles 2 and 3 respectively. Therefore, "ACT1" is executed in cycle 1 and 3. However, in cycle 4 there is no rising edge for the Action Control, thus, "ACT1" is not executed again.

"L" Qualifier

"L" Qualifier



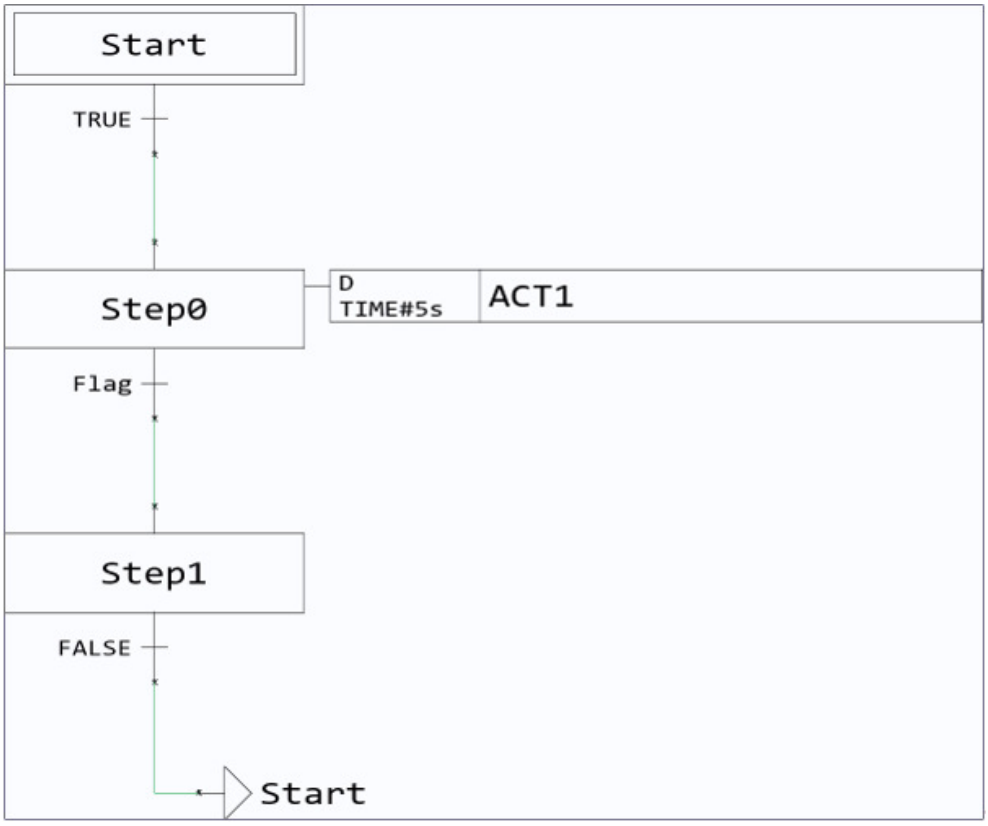
Consider the above example with "Flag" variable set to FALSE initially. In cycle 0, Step "Start" is active and no Action is executed. In cycle 1, "Step0" becomes active, the timer is started and "ACT1" Action is executed. In the consecutive cycles, the program state stays at "Step0" being active, therefore the timer is not reset and keeps counting. Then …

- if "Flag" variable stays FALSE or becomes TRUE after 5 seconds has passed, in each cycle, until the time limit of 5 seconds has passed "ACT1" Action is executed. The Action is then executed one last time during the Final Scan in the next cycle.
- if "Flag" variable becomes TRUE before 5 seconds has passed, then after the current execution is completed, the Action is executed one last time during the Final Scan in the next cycle.

"D" Qualifier

Programming

"D" Qualifier



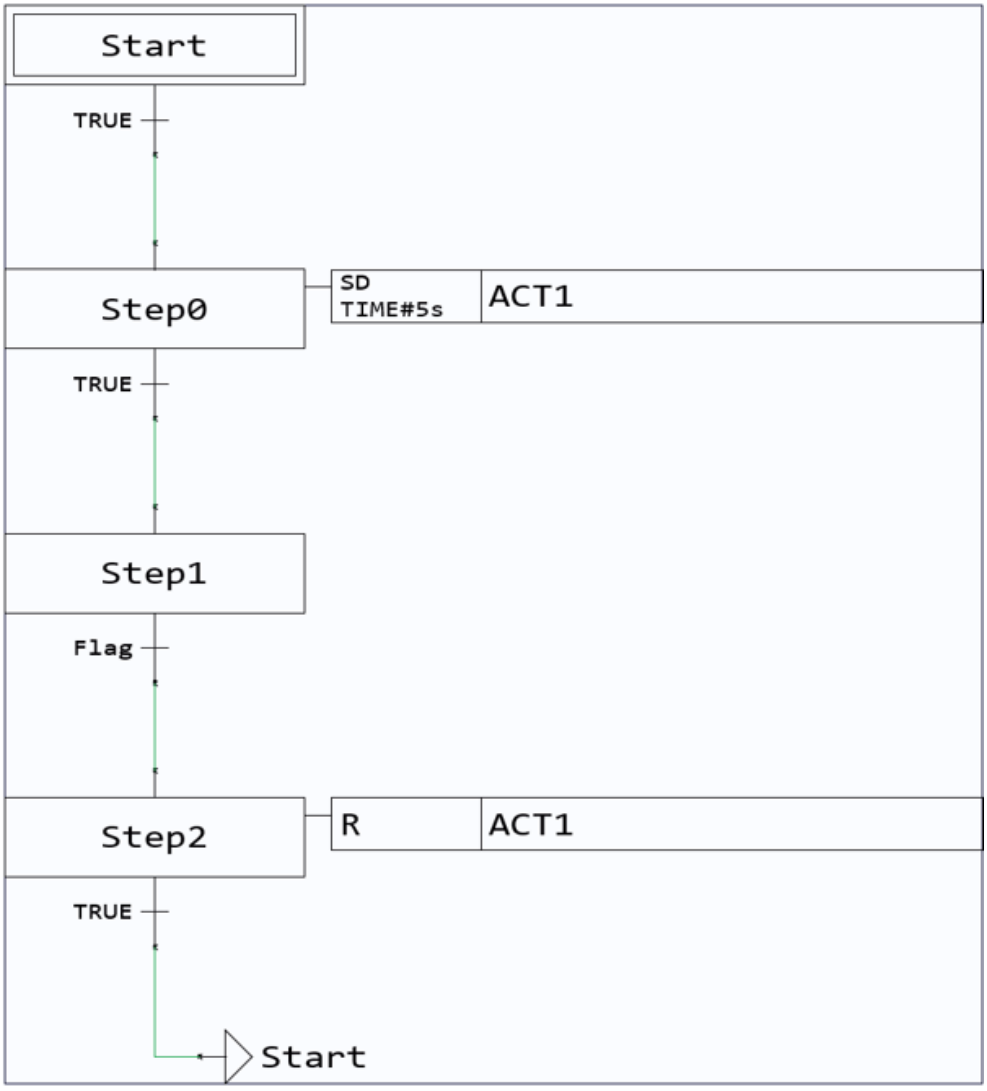
Consider the above example with "Flag" variable set to FALSE initially. In cycle 0, Step "Start" is active and no Action is executed. In cycle 1, "Step0" becomes active, the timer is started and once the specified time of 5 seconds has passed, "ACT1" Action is executed. Then, the Action is executed again in the Final Scan in the next cycle.

If "Flag" variable becomes TRUE and the Step is deactivated before 5 seconds has passed, the Action will not be executed. The Action will also not be executed in the Final Scan in this case.

"SD" Qualifier

Programming

"SD" Qualifier

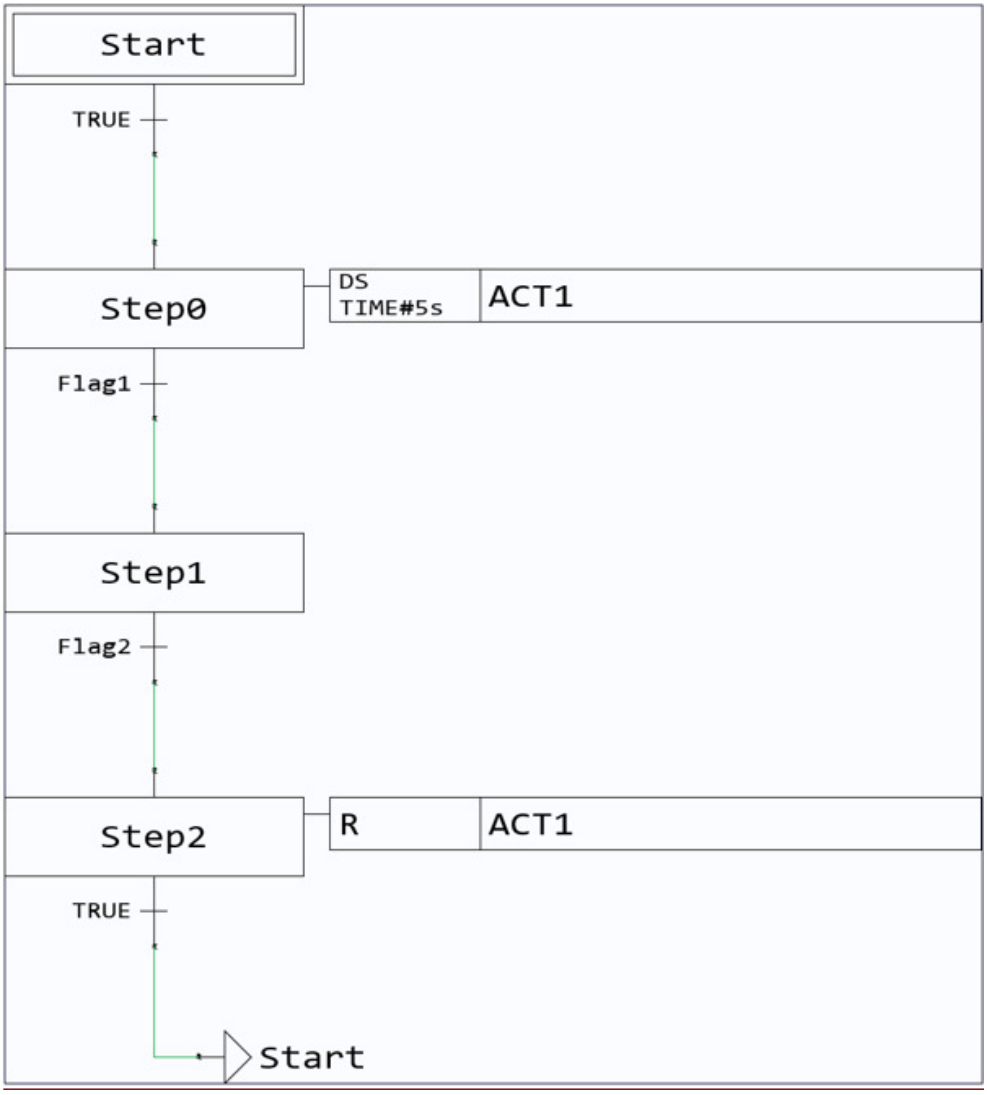


Consider the above example with "Flag" variable set to FALSE initially. In cycle 0, Step "Start" is active and no Action is executed. In cycle 1, "Step0" becomes active, and the timer is started. In the next cycle, "Step1" is activated and stays active in the subsequent cycles. After 5 seconds have passed, even though "Step0" is not active, "ACT1" is executed and keeps executing in the next cycles. Once the "Flag" variable becomes TRUE, "Step2" is activated, "ACT1" is executed one last time and "R" Qualifier deactivates the Action so "ACT1" is no longer executed. If the "Flag" variable became TRUE before 5 seconds delay elapsed, then "ACT1" would not be executed at all.

"DS" Qualifier

Programming

"DS" Qualifier



Consider the above example with "Flag1" and "Flag2" variables set to FALSE initially. In cycle 0, Step "Start" is active and no Action is executed. In cycle 1, "Step0" becomes active, and the timer is started. Then …

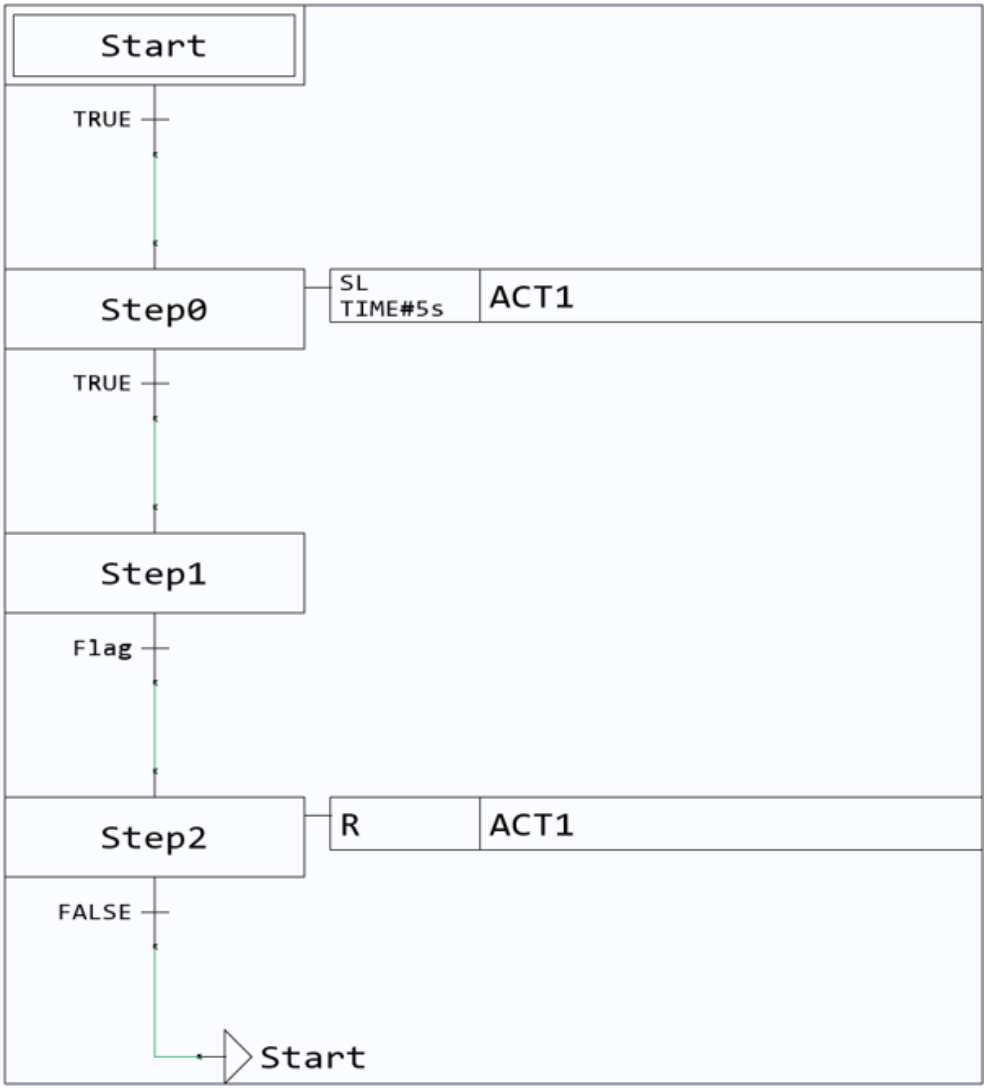
- if "Flag1" becomes TRUE before 5 seconds elapsed, then "Step1" is activated. Since "Step0" is no longer active, when 5 seconds pass the Action will not be executed.
- if "Flag1" stays FALSE, since "Step0" is still active, when 5 seconds pass the Action will be executed and continue to be executed in the subsequent cycles. Afterwards, if "Flag1" becomes TRUE, the Action continues to be executed even though "Step0" is no longer active.

Once the "Flag2" variable becomes TRUE, "Step2" is activated, "ACT1" is executed one last time and "R" Qualifier deactivates the Action so "ACT1" is no longer executed.

"SL" Qualifier

Programming

"SL" Qualifier



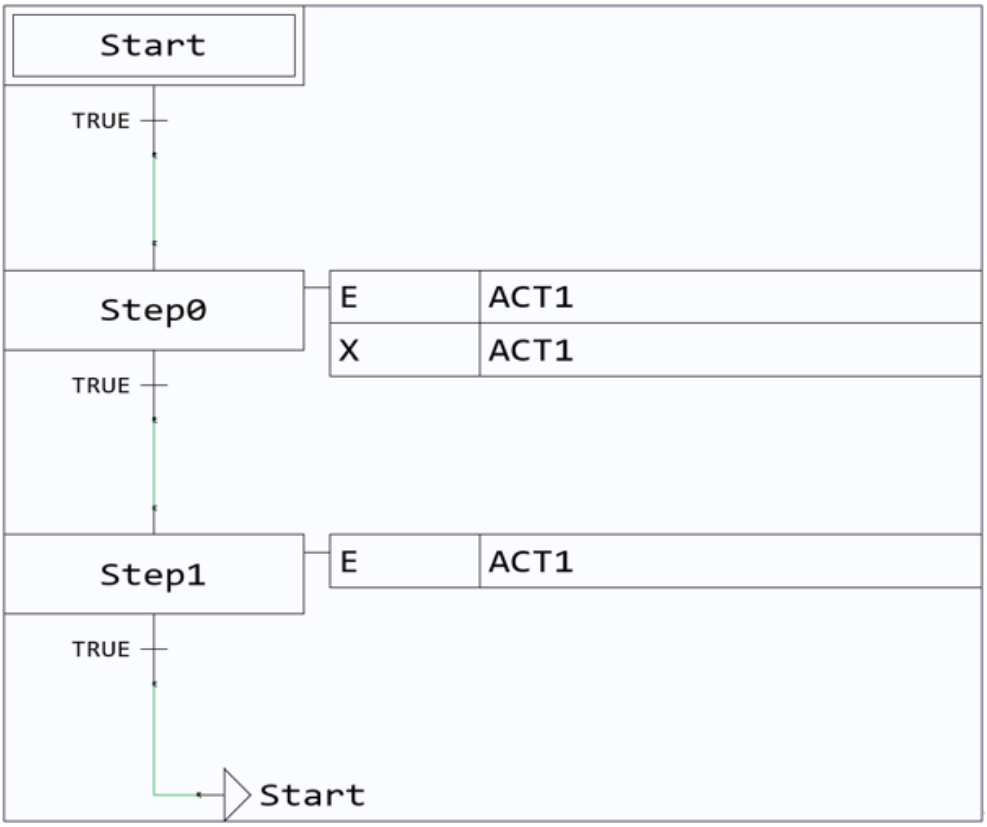
Consider the above example with "Flag" variable set to FALSE initially. In cycle 0, Step "Start" is active and no Action is executed. In cycle 1, "Step0" becomes active, the timer is started and "ACT1" Action is executed. In cycle 2, "Step1" becomes active and although "Step0" is inactive the Action is still executed and continues to be executed in next cycles as long as 5 seconds has not elapsed. If …

- "Flag" becomes TRUE before 5 seconds has passed, "Step2" is activated, "ACT1" is executed one last time and "R" Qualifier deactivates the Action so "ACT1" is no longer executed.
- "Flag" becomes TRUE after 5 seconds has passed, "Step2" is activated but "R" Qualifier has no effect on the Action as it has already stop being executed.

"E" and "X" Qualifiers

Programming

"E" and "X" Qualifiers



Loop Index	Start	ACT1
0	Start	Executed once as "Step1" entry Action
1	Step0	Executed twice: • once as "Step0" exit Action • once as "Step1" entry Action
2	Step1	Not executed

About Global Variables

It is possible to add global variables to a PLC unit.

It is recommended that developers consult applicable coding guidelines before adding any global variables.

C Code in PLUS+1 GUIDE

PLUS+1® GUIDE supports the following methods of including C code in a project:

- C code POUs
- C code files

C code POUs are regular POUs, except the implementation part of the POU is written in C.

C code files are regular .h and .c files.

Programming

Warning

Using untested C code POUs and/or C code Files in an application can produce an application that is unstable and unpredictable. An unstable, unpredictable application can cause unexpected machine movements that result in personal injury and equipment damage. To reduce the risks in using an application that contains C code POUs and/or C code Files, developers must be certain that the code:

- Operates with its intended hardware correctly and without defects
- Does not contain code with lengthy execution times that can cause the controller to significantly slow down or freeze
- Does not contain code with potential endless loops and "wait until" statements
- Does not interfere with the data flow principles of GUIDE code
- Does not access the hardware application programming interface or any other platform resources such as the RAM, EEPROM, CPU registers, and kernel
- Accesses only the memory allocated within the C code itself
- Keeps the use of stack memory to a minimum

General Considerations Regarding C Code in a PLUS+1 GUIDE Environment

About Compatibility

GUIDE graphical code (as well as standard PLC61131-3 code) that compiles in one version of GUIDE will generally also compile in later GUIDE versions without modifications. C code in the form of C code POUs and C code Files is more volatile in this respect.

GUIDE will as far as possible try to make sure that C code that compiles in one version in GUIDE should also compile in following versions, but in some cases, minor adjustments may be necessary.

For example, the handling of VAR_IN_OUT parameters in C code POU Function Blocks was changed from GUIDE 9.0 to 9.1 due to underlying changes in the code generation that aimed to improve performance and standard compliance.

It is expected that C code POUs will be more prone to updates than C code Files, so from that perspective it could make sense to use the C code POUs mainly as thin wrappers for C code Files.

Accessing C Code Generated by PLUS+1 GUIDE from C Code POUs or C Code Files

Do not attempt to access any tool-generated C code from C code POUs or C code Files. The tool-generated C code may change from one compilation to the next, or when changing HWD files, or when switching to another PLUS+1® GUIDE version, or depending on other differences in the configuration. This means, among other things, that no POU should be called from C code POUs or from C code Files.

PLUS+1® GUIDE will check that no generated header file is included from C code POUs or from C code Files before compiling the code.

Validity of calls between different types of code

Caller	Call of	Is call type valid
Graphical GUIDE code module	Graphical GUIDE code module	Yes
Graphical GUIDE code module	Standard POU (ST/FBD/LD/SFC/IL)	Yes
Graphical GUIDE code module	C code POU	Yes
Graphical GUIDE code module	C code File	Not possible
Standard POU (ST/FBD/LD/SFC/IL)	Graphical GUIDE code module	Not possible
Standard POU (ST/FBD/LD/SFC/IL)	Standard POU (ST/FBD/LD/SFC/IL)	Yes
Standard POU (ST/FBD/LD/SFC/IL)	C code POU	Yes
Standard POU (ST/FBD/LD/SFC/IL)	C code File	Not possible
C code POU	Graphical GUIDE code module	Not allowed

Programming

Validity of calls between different types of code (continued)

Caller	Call of	Is call type valid
C code POU	Standard POU (ST/FBD/LD/SFC/IL)	Not recommended[1]
C code POU	C code POU	Not recommended[1]
C code POU	C code File	Yes
C code File	Graphical GUIDE code module	Not allowed
C code File	Standard POU (ST/FBD/LD/SFC/IL)	Not recommended[1]
C code File	C code POU	Not recommended[1]
C code File	C code File	Yes

[1] Code may not work without modifications in future versions of GUIDE.

Supported HWDs

The C in GUIDE feature is available for any PLUS+1® HWD that supports PLC code. Specifically, that means any HWD which supports any of the following CCP types:

- CCP_TI_v525
- CCP_TI_v649
- CCP_SGL_2010Q1202
- CCP_ARM_THUMB_v410
- CCP_ARM_v410
- CCP_STM32FX0X441
- CCP_STM32FX0X474

The easiest way to check this is to see if the Call POU component is available in the **Component** tree under **Connection**.

`CHAR_BIT`

When developing C code for PLUS+1® hardware, it is critical to understand the meaning of `CHAR_BIT` which is defined in `<limits.h>`.

At the time of writing, all PLUS+1® hardware use compilers where the value of `CHAR_BIT` is defined as either 8 or 16, but this is not guaranteed to always be the case. Types such as `uint8_t` and `int8_t` won't be available when `CHAR_BIT` has a value of 16.

There are at least 3 ways to handle this:

- Avoid using types such as `uint8_t` and `int8_t` completely
- Types such as `uint_least8_t` and `int_least8_t` might be used instead
- Check the value of `CHAR_BIT` and use different implementations depending on its value, see sample code below.

```
#if CHAR_BIT == 8

    //uint8_t might be used here

#elif CHAR_BIT == 16

    //uint8_t can't be used here

#else

    #error "Unsupported CHAR_BIT value!"

#endif
```

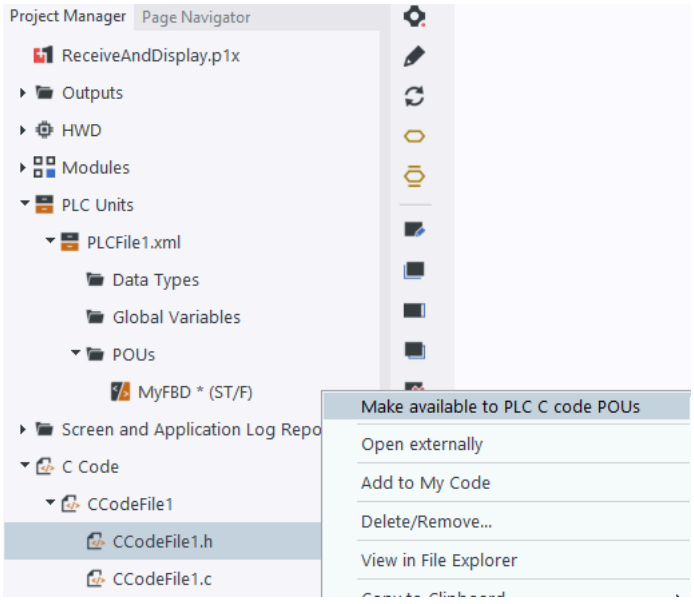
Programming

#include directives

The #include directive should only be used in C code Files, never in C code POUs.

C code POUs will instead have access to all header files that have been made available to PLC code using the settings provided in the **Project Manager** tree in GUIDE.

All available headers are listed under the **Project Manager** > **PLC Units**.



C code Files may include any C99 header files, and also any header files which have been added to the **Project Manager** tree in GUIDE. Note that header files which are considered Temporary Project Files may not be added to the **Project Manager** tree in this way. Including any other header file is considered an error and will prevent compilation.

About C Data Types

It is recommended to use the data types defined by the C99 standard header file: **<stdint.h>**

To see how some common C data types maps to GUIDE and IEC61131 data types, see the following table:

PLUS+1® GUIDE, IEC61131 and C Data Types

PLUS+1® GUIDE Data Type	IEC61131 Data Type	C Data Type
BOOL	BOOL	Depends on CHAR_BIT, either uint8_t or uint16_t
U8	USINT, BYTE	Depends on CHAR_BIT, either uint8_t or uint16_t
S8	SINT	Depends on CHAR_BIT, either int8_t or int16_t
U16	UINT, WORD	uint16_t
S16	INT	int16_t
U32	UDINT, DWORD	uint32_t
S32	DINT	int32_t
U64	ULINT, LWORD	uint64_t
S64	LINT	int64_t
F32	REAL	float
F64	LREAL	double
ARRAY[n]Type	ARRAY[0..n-1] OF Type	Type[n]

Programming

It is not recommended to use GUIDE data type names in user defined C code Files and C code POUs.

It is not recommended to use IEC61131 data type names in user defined C code Files. (In C code POUs, IEC61131 data type names are mandatory in the interface part, but should not be used in the implementation part.)

In cases where the exact bit count is less important, it might be preferable to use types such as uint_least8_t, or uint_fast8_t. In such cases, care must be taken to make correct conversions when passing data over the GUIDE/IEC61131 interface.

C Code POUs

C code POUs can be called directly from graphical GUIDE code using the same component as for any POU, the **Call POU** component. They can also be called from other POUs written in any of the standard languages.

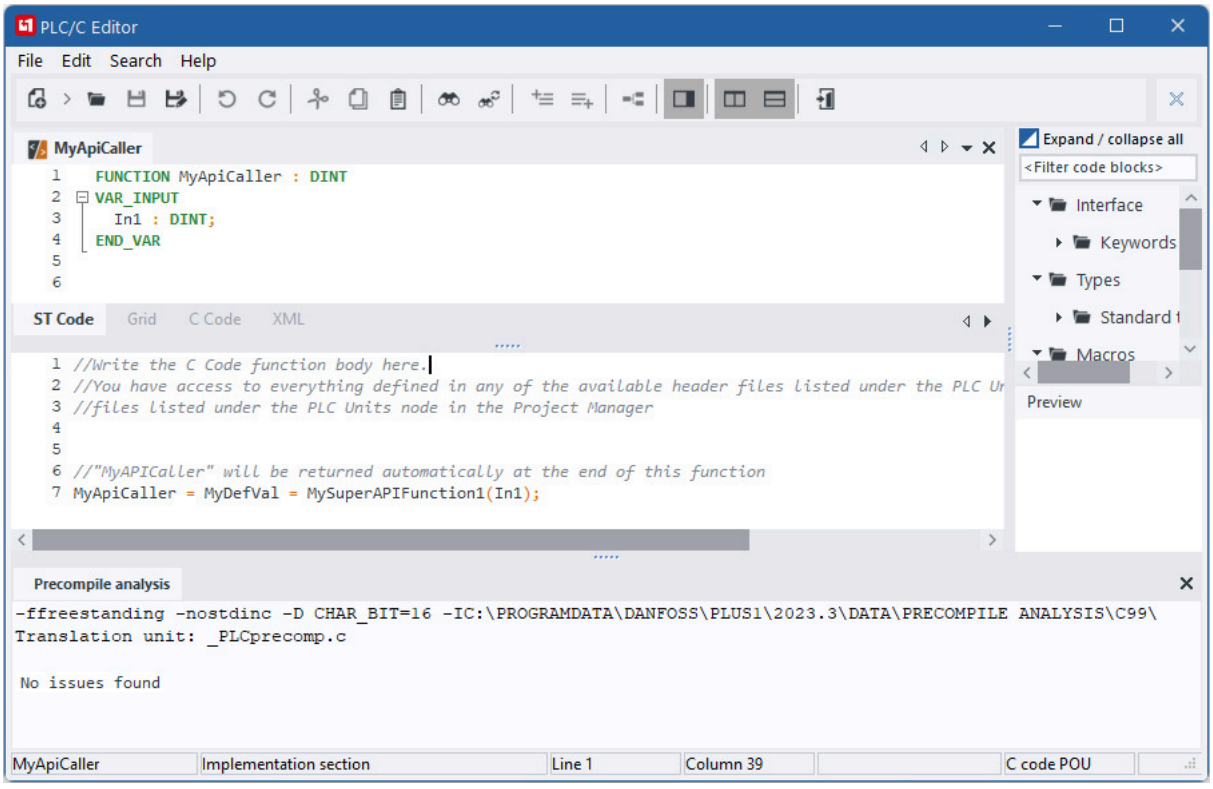
The main use case for POUs developed with C code is to call regular C code functions defined in .c and .h files that have been added to the project. It is also possible to implement the whole intended functionality within the C code POU itself.

The C code is written directly in the implementation section of the POU as usual.

As for the other POU languages, the function delimiters (For example, `END_FUNCTION` for ST code) should be omitted in the implementation section. That is, the pair of outer '{', '}'- brackets of the generated C code function are added automatically by the tool.

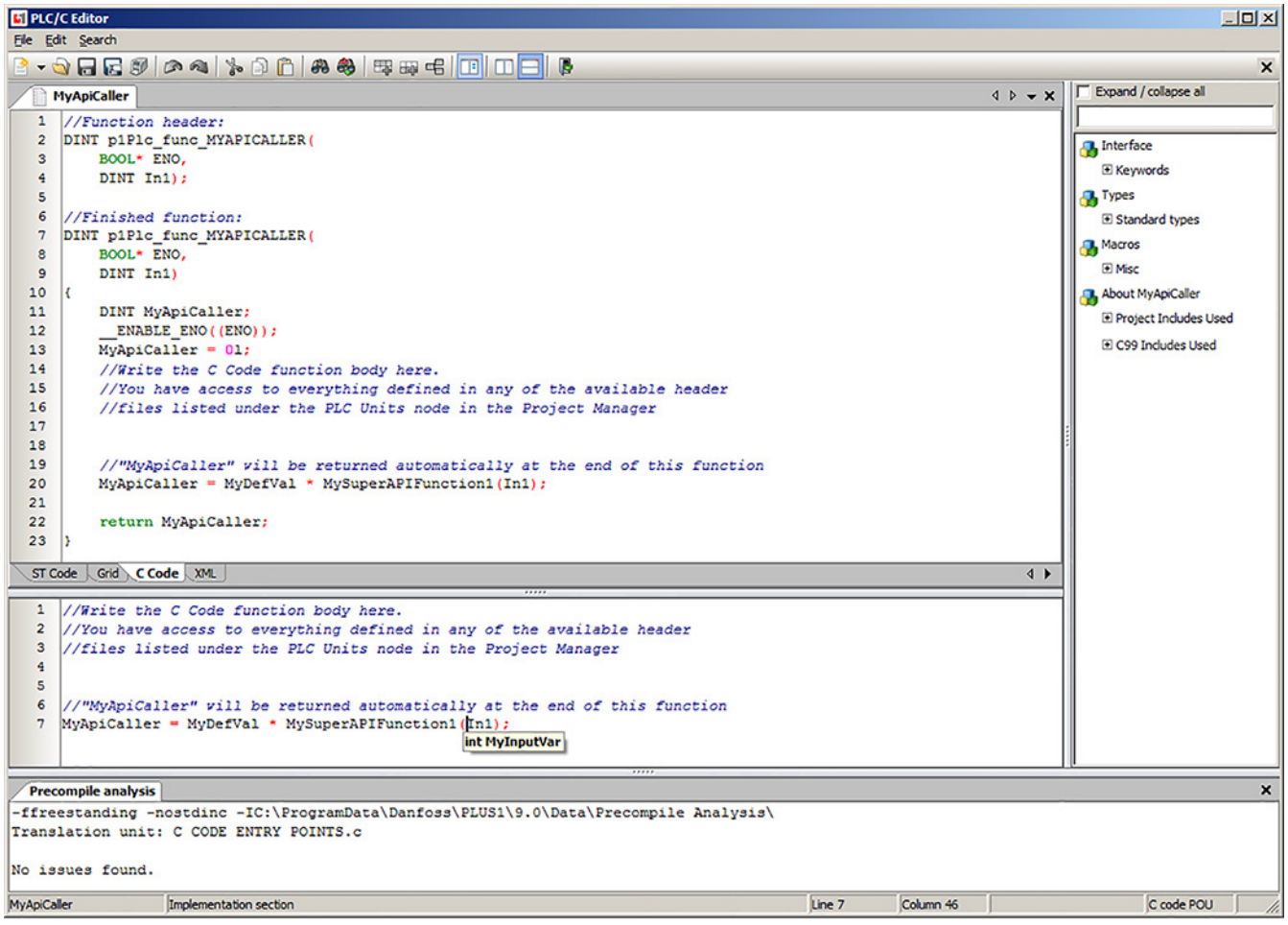
C code POUs will also contain automatically generated code to initialize temporary variables before the user defined code is run. It is therefore recommended not to initialize such variables in user written code since doing so would be redundant.

Furthermore, C code POUs for functions will also contain automatically generated code to ensure that the function return value will not be undefined. Functions will have a local variable defined by the same name as the POU, and the recommendation is to assign the intended return value to that variable, rather than manually using the return statement in user written code.



Programming

The tab named **C code** in the interface section can be used to see what the generated end result C code POU will be.



How to access POU parameters in a C code POU

POU parameters	C code access for FUNCTIONs	C code access for FUNCTION_BLOCKs

Inputs:
`VAR_INPUT`
`  In1 : DINT;`
`  ArrIn1 : ARRAY[0..1] OF DINT;`
`END_VAR`
	In1
	ArrIn[0]
	ArrIn[1]
		self->In1
		self->ArrIn[0]
		self->ArrIn[1]

Outputs:
`VAR_OUTPUT`
`  Out1 : DINT;`
`  ArrOut1 : ARRAY[0..1] OF DINT;`
`END_VAR`
	*Out1
	ArrOut1[0]
	ArrOut1[1]
		self->Out1
		self->ArrOut1[0]
		self->ArrOut1[1]

Programming

How to access POU parameters in a C code POU (continued)

POU parameters	C code access for FUNCTIONs	C code access for FUNCTION_BLOCKs

In/out:
`VAR_IN_OUT`
`  Io1 : DINT;`
`  ArrIo1 : ARRAY[0..1] OF DINT;`
`END_VAR`
	*Io1
	ArrIo[0]
	ArrIo[1]
		*Io1
		ArrIo[0]
		ArrIo[1]

Internal:
`VAR`
`  Var1 : DINT;`
`  ArrVar1 : ARRAY[0..1] OF DINT;`
`END_VAR`
	Var1
	ArrVar1[0]
	ArrVar1[1]
		self->Var1
		self->ArrVar1[0]
		self->ArrVar1[1]

Temporary internal:
`VAR_TEMP`
`  Tmp1 : DINT;`
`  ArrTmp1 : ARRAY[0..1] OF DINT;`
`END_VAR`
	N/A	Tmp1
		ArrTmp1[0]
		ArrTmp1[1]

Scalar return value: `FUNCTION MyFunc : DINT`	MyFunc	N/A
Array return value: `FUNCTION MyFunc : ARRAY[0..1] OF DINT`	MyFunc[0] MyFunc[1]	N/A

C Code Files

C code Files are regular **.h** and **.c** files that have been added to a project.

Functions defined in C code Files can only be called from C code POUs, they cannot be called directly from Graphical GUIDE code module.

C code POUs have access to all functionality of those header files that have been selected to be available for PLC code. By default, header files will not be made available to PLC code.

To avoid potentially conflicting definitions, it is recommended to make as few headers as possible available to PLC code.

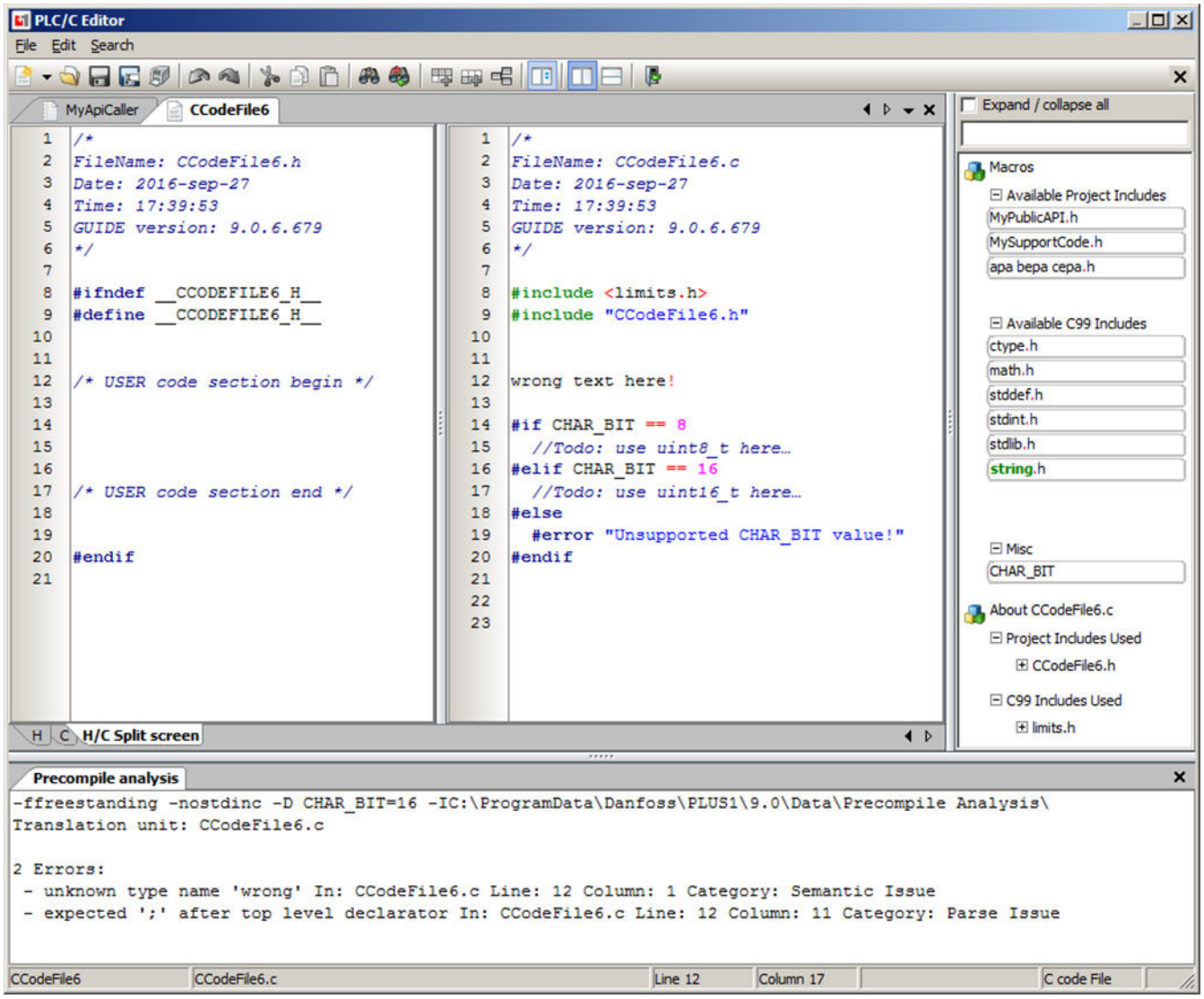
Precompile Analysis

Automatic error checking of .c and .h files is performed on save, and the result is presented in the “Precompile analysis” tab located at the bottom of the editor.

Programming

The precompile analysis is performed based on the rules and implementation of a specific C compiler, and will not necessarily match the rules and implementation of the actually used C compiler for the target system.

It is not recommended to attempt to compile a project before fixing every reported warning and error of the Precompile analysis.



C++ Code in PLUS+1 GUIDE

PLUS+1® GUIDE supports the following methods of including C code in a project:

- C++ code Files

C++ code Files are regular `.h` and `.cpp` files.

Legacy `.hpp` files are also supported, but not recommended.

Programming

Warning

Using untested C++ code Files in an application can produce an application that is unstable and unpredictable. An unstable, unpredictable application can cause unexpected machine movements that result in personal injury and equipment damage. To reduce the risks in using an application that contains C++ code Files, developers must be certain that the code:

- Operates with its intended hardware correctly and without defects
- Does not contain code with lengthy execution times that can cause the controller to significantly slow down or freeze
- Does not contain code with potential endless loops and “wait until” statements
- Does not interfere with the data flow principles of GUIDE code
- Does not access the hardware application programming interface or any other platform resources such as the RAM, EEPROM, CPU registers, and kernel
- Accesses only the memory allocated within the C code itself
- Keeps the use of stack memory to a minimum

About Compatibility

GUIDE graphical code (as well as standard PLC61131-3 code) that compiles in one version of GUIDE will generally also compile in later GUIDE versions without modifications. C+ code in the form of C++ code Files is more volatile in this respect.

GUIDE will as far as possible try to make sure that C++ code that compiles in one version in GUIDE should also compile in following versions, but in some cases, minor adjustments may be necessary.

It is expected that C code POUs will be more prone to updates than C++ code Files, so from that perspective it could make sense to use the C code POUs mainly as thin wrappers for C++ code Files.

Accessing C Code Generated by PLUS+1 GUIDE from C Code POUs or C Code Files

Do not attempt to access any tool-generated code from C++ code Files. The tool-generated code may change from one compilation to the next, or when changing HWD files, or when switching to another PLUS+1 GUIDE version or depending on other differences in the configuration. This means, among other things, that no POU should be called from C++ code Files.

PLUS+1 GUIDE will check that no generated header file is included from C++ code Files before compiling the code.

Validity of calls between different types of code

Caller	Call of	Is call type valid
Graphical GUIDE code module	C++ code File	Not possible
Standard POU (ST/FBD/LD/SFC/IL)	C++ code File	Not possible
C code POU	C++ code File	Yes
C code File	C++ code File	Yes
C++ code File	Graphical GUIDE code module	Not allowed
C++ code File	Any POU	Not recommended[1]
C++ code File	C code File	Yes
C++ code File	C++ code File	Yes

[1] Code may not work without modifications in future versions of GUIDE.

Supported HWDs

The C++ in GUIDE feature is only available for a subset of PLUS+1® HWDs.

Programming

Check the HWD documentation for information about whether your HWD supports C++ or not.

C++ Code Files and how to call them

C++ code Files are regular `.h` and `.cpp` files that have been added to a project.

Legacy `.hpp` files are also supported, but not recommended in GUIDE.

The C++ Core Guidelines by Bjarne Stroustrup and Herb Sutter suggest to use `.h` and `.cpp` file extensions for best interoperability with C code.

Functions defined in C++ code Files can only be called from C code POUs, C code files, and other C++ code files. They cannot be called directly from Graphical GUIDE code module.

C code POUs have access to all functionality of those header files that have been selected to be available for PLC code. By default, header files will not be made available to PLC code.

C++ header files should usually contain an ifdef-section like:

```
#ifdef __cplusplus
extern "C" {
#endif

//C code interface goes here...

#ifdef __cplusplus
}
#endif
```

Calling a C++ function from a C code POU

1. Make the header file available to POU code via context menu click on the header file.
2. C code POUs can now call any function declared inside the extern “C” section of the header file.

Calling a C++ function from a C code file

1. Write an #include directive in the C code file to include the header file.
2. This C code file can now use the functions declared inside the extern “C” section of the header file.

Calling a C++ function from a C++ code file

1. Write an #include directive in the C++ code file to include the header file.
2. This C++ code file can now use everything declared inside the header file.

Dynamic memory allocation

Dynamic memory allocation is not recommended due to the risk of memory exhaustion, leaks or fragmentation in an embedded system. Avoid the operator new.

Exceptions

The use of exceptions is not recommended. Consider that in a GUIDE application, any C++ code will at some point be called from C code. So error handling for the application as a whole cannot be done consistently with just exceptions. Any use of exceptions in the C++ portion of the application code would also need to be caught at every C++ function that has a C interface and can be called from C code to avoid the exception from reaching the calling C code.

Programming

Programming Tips and Tricks

Auto-completion

Auto-completion for C and ST code is triggered by pressing **Ctrl** > **Space**, and will then present a list of possible completions based on the characters to the left of the cursor.

For C code, all identifiers that are available in the current scope, and which starts with the character sequence will be available in the list. If the text preceding the cursor indicates a struct variable and ends with '.' or '->', then the list will instead be filled by the members or the struct. All .c and .h files in the current translation unit will be considered. The first time after adding a .c file to a project, it is necessary to perform a "save all" before the auto-complete feature becomes active in that file.

For ST code auto-completion works similarly, but additional information about the completion items is also provided, such as whether they represent functions or variables.

Graphical PLC code does not have auto-completion in the same way as textual code, but in most cases the Query dialog for graphical PLC Elements will provide drop-down lists which contain relevant selections for the current scope.

Parameter Hints

For both C and ST code, it is possible to see a hint that describes the needed parameters for a function call.

To activate this feature, simply place the cursor between the parenthesis of the function call.

Jump to Definition

In ST code, it is possible to jump to the definition of a function or function block simply by holding down the **Ctrl** key and then clicking on the function name.

Issue Indicators

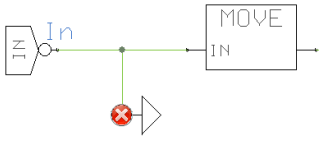
PLUS+1® GUIDE provides a system of graphical indicators to pinpoint important places in the code, such as:

error	if a code statement is not correct
warning	if a code statement is not safe or/and compliant with the standard
hint	if it is possible to make code safer/faster
search hit	if a search query succeeded and a location of a searched item can be pin-pointed.

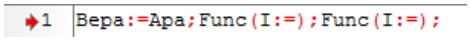
All of the above four indicators are available for GUIDE code as well as for the IEC 61131-3 languages. (Structured text, Function Block Diagram and Ladder Logic).

The following overview lists supported indicator images:

Error graphical code (GUIDE + FBD/LD POUs)

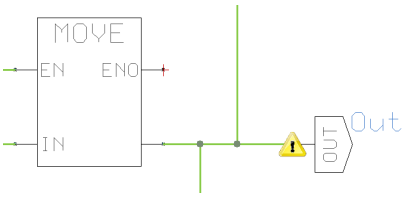


Error text code (ST POUs)



Programming

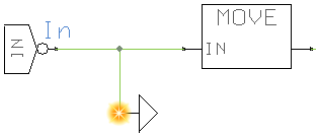
Warning graphical code (GUIDE + FBD/LD POUs)



Warning text code (ST POUs)

```
1  //warning
2  Bepa:=Apa;Func(I:=);Func(I:=);
```

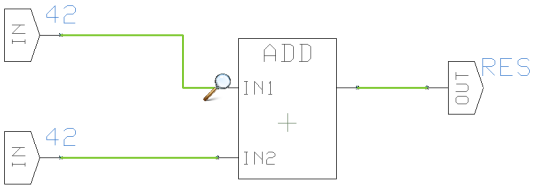
Hint graphical code (GUIDE + FBD/LD POUs)



Hint text code (ST POUs)

```
1  Bepa:=false;
```

Search hit graphical code (GUIDE + FBD/LD POUs)



Search hit text code (ST POUs)

```
1  Bepa:=Cepa;
```

Screen Editors

Learn more about the Vector-Based and Classic Screen Editors offered by PLUS+1® GUIDE.

The PLUS+1® GUIDE has these two screen editors:

- Vector-Based Screen Editor, which you use with the **Show Screen** component and **Screen Definitions** to create an application for a PLUS+1® graphical terminal.

  For more information, see *Vector-Based Screen Editor* on page 450.
- Classic Screen Editor, which you use with the **Define Areas Page** and the **Define Screen Pages** components to create an application for a PLUS+1® graphical terminal.

  See *Classic Screen Editor*.

The **Define Areas Page** and the **Define Screen Pages** components are hardware-dependent components.

They become available after you install a hardware file (HWD) in the **Project Manager** tab that supports the graphical terminal for which you are creating an application.

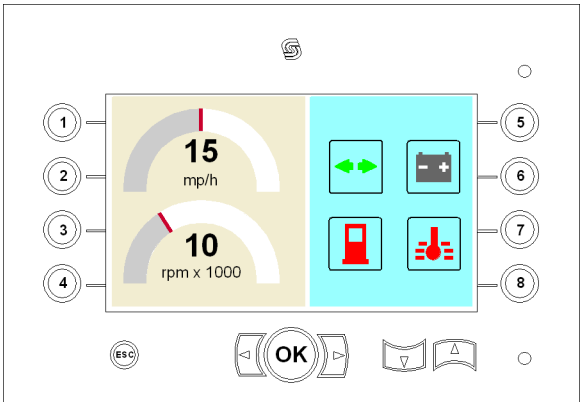
Classic Screen Editor

The Classic Screen Editor consists of a **Define Areas Page** and **Define Screens Page**. You use these pages to define:

- Screen areas that appear in a graphical terminal.
- Contents (such as images and text) of each Screen Area.

The **Define Areas Page** and the **Define Screen Page** are available in the **Components** tab of the PLUS+1 GUIDE software. You drag these pages into the **Application** of a graphical terminal template when creating a graphical application.

Typical Graphical Terminal



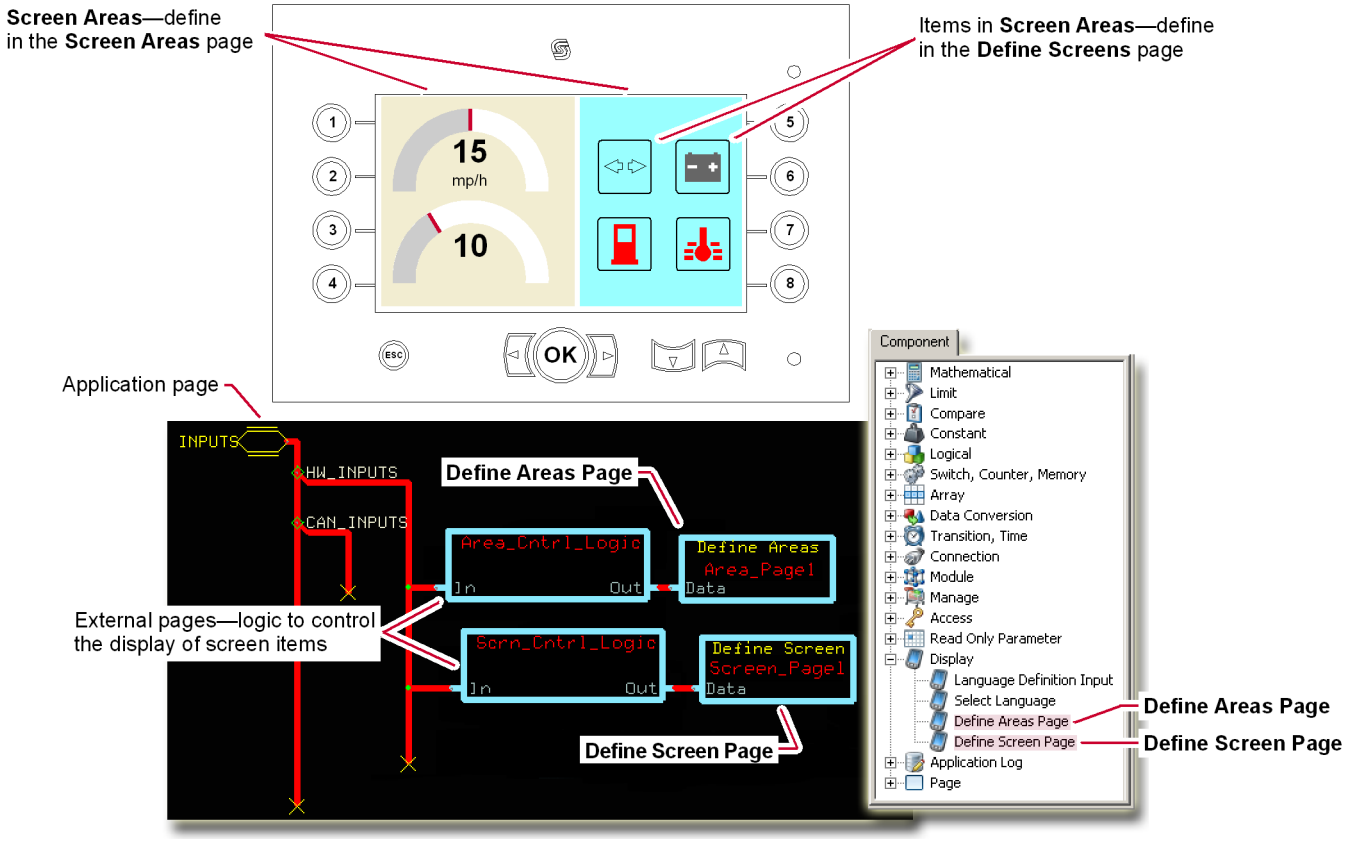
The Classic Screen Editor define the:

- Tan and aqua Screen Areas.
- Bitmap images and text that appears in each Screen Area.

The Classic Screen Editor can manipulate (rotate, turn on and off) images but it cannot create them or size them to fit in a graphical terminal. To create images, you need a pixel-editing program such as Adobe® Photoshop® Elements software.

Screen Editors

Classic Screen Editor Elements



Callouts in this figure identify the basic elements in a typical graphical terminal screen display, **Define Areas Page** and the **Define Screen Page** that define these elements.

You place these pages in the **Application** of a graphical terminal template and external pages with the logic that controls the display of screen items.

Application Page

You create:

Screen Editors

- Most of an application for a graphical terminal in this page. The template that you use must match the graphical terminal for which you are creating an application.
- A graphical terminal application using a **Define Screen Page** and a **Define Areas Page**.

  Select these pages in the **Components** tab and drag them into the **Application** page. Put additional control logic in external pages, as needed.

  The **Inputs** bus brings signals from graphical terminal buttons and other sources to these pages.

Define Areas Page

- Use the tools that display when you enter this page to create the Screen Areas that appear on the screen of your graphical terminal. A Screen Area is the background against which images, lines, and text appear.
- Use the tools in the **Define Areas Page** to define the position of each Screen Area, as well as the Screen Area's size and color. A Screen Area can fill all or part of a graphical terminal's screen.
- After you bring signals to the Define Areas Page, use the editing tools within this page to select the signals that enable the display of Screen Areas. The Define Areas Page does not support sub-buses. Always use a main bus to bring signals to this page.

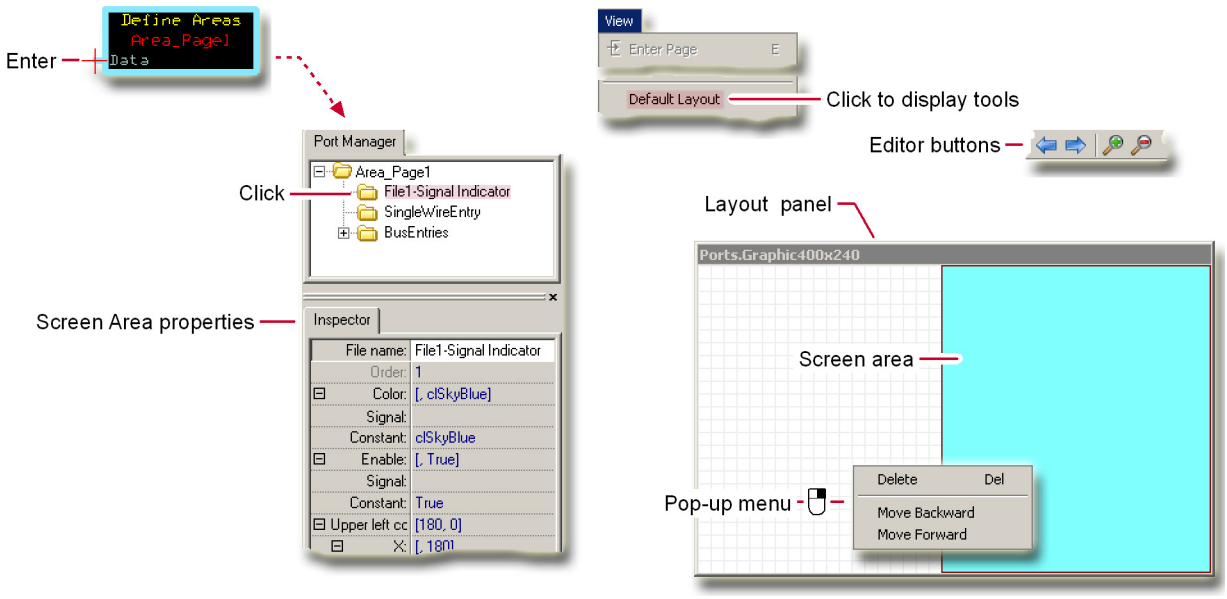
  The **File menu's** > **Import** commands do not work with this component. Importing this component into a new project strips the component of its contents.

Define Screen Page

- Use the tools that display when you enter this page to define the contents of each Screen Area. The contents of each Screen Area include bitmap images, lines, and text. (Before you can define the contents of a Screen Area, you must first use the **Define Areas Page** to create the Screen Area.)
- Use the tools in the **Define Screen Page** to define the appearance and position of each item that appears in a Screen Area.
- After you bring signals to the **Define Screen Page**, use the editing tools on this page to select the signals that control the behavior of items in Screen Areas.
- Use signals to show and hide items, move and rotate items, and display changing numeric values.

Screen Editors

Define Areas Page



The **Define Areas Page** defines Screen Areas. A Screen Area is the background or area in which images, lines, and text appear. The preceding figure shows the tools (such as tabs, panes, and buttons) within the **Define Areas Page** that are used to create the Screen Areas. These tools display when you first enter the **Define Areas Page**. (The **Screen Library** tab also displays but is not used when creating Screen Areas. The figure here does not show this tab.)

The **File** menu's **Import Page** and **Import Block** commands do not work with the **Define Areas** block. Importing a **Define Areas Page** into a new project strips this page of its contents. The **Define Areas Page** does not support sub-buses. Always use a main bus to bring signals to this page.

Define Areas Page

Item	Description
Port Manager tab	Use this tab to manage Screen Areas. **Areas**—click for a tree view of all the Screen Areas. Click a Screen Area name to see its properties in the **Inspector** tab and to move the area to the front of the Layout tab. **SingleWireEntry**—not used here. **BusEntries**—click for a tree view of all the signals available on the **Define Areas Page**'s **Data** bus port.
Inspector tab	Use this tab to manage the properties of Screen Areas.
Layout tab	Use to layout Screen Areas. Drag as needed to resize and reposition Screen Areas placed in this tab. When you click a Screen Area in this tab, the Area moves to the front, the **Port Manager** tab highlights its name, and the **Inspector** tab shows its properties. Right-click a Screen Area to open a pop-up menu with **Delete**, **Move Backward**, and **Move Forward** commands. Use the **Move Backward** and **Move Forward** commands to change the **Inspector** tab's **Order** property.
Editor buttons/keys	The size of your monitor determines the row in which these buttons appear.
 ——	Move backward and forward through your page navigation history.

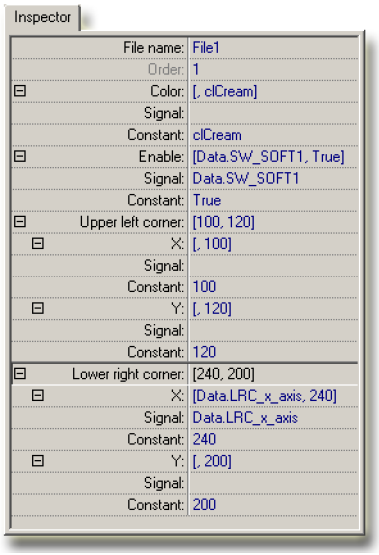
Screen Editors

Define Areas Page (continued)

Item		Description
	PgDn PgUp	Zoom out and in on the Layout tab.
	L	Exit the Classic Screen Editor and return to the Application page.

Define Areas Page—Inspector Tab

Use this tab to manage the properties of Screen Areas. To view the properties of a Screen Area, click the Screen Area name listed in the **Port Manager** tab or click the Screen Area in the Layout tab.



Inspector tab

Item	Description
File name	The name of the Screen Area.
Order	Indicates the display order of a Screen Area when several Screen Areas simultaneously have an **Enable** property that has a True **Entry** signal or a **Value** that is **True**. The order in which you assign Screen Areas in the **Port Manager** tab initially sets the front-to-back **Order** property. • A Screen Area with an **Order** property of **1** displays in front of a Screen Area with an **Order** property of **2**; • A Screen Area with an **Order** property of **2** displays in front of a Screen Area with an **Order** property of **3**. To change the **Order** property, right-click a Screen Area. In the pop-up menu that opens, click the **Move Forward** or **Move Backward** command. A signal-enabled Screen Area always displays in front of a True-enabled Screen Area.
Color	• **Signal**—click to select a signal to control the Screen Area color. • **Constant**—click to see a list of standard colors. Double-click to open a standard Windows palette from which you can select a color.

Screen Editors

Inspector tab (continued)

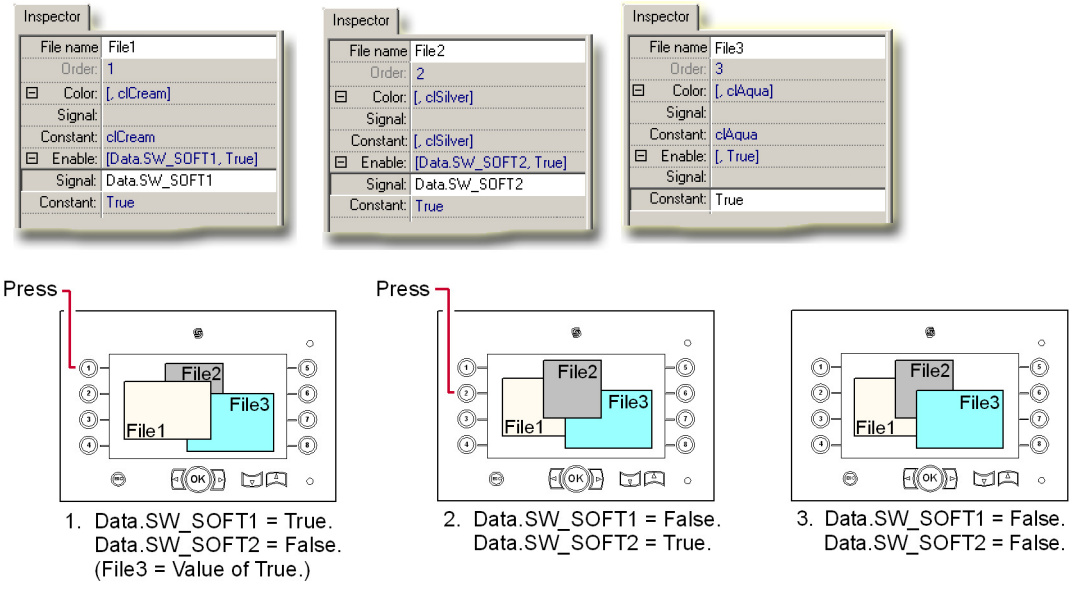
Item	Description
Enable	Enables the display of a Screen Area. • **Signal**—click to select a signal from the drop-down list. A True signal brings this screen to the front of the display tab. A True signal makes the results of changed **Color**, **Upper left corner**, and **Lower right corner** properties visible. • **Constant**—click **True** or **False**. — A **True** Screen Area displays in front until a signal-enabled Screen Area receives a True signal. — A **False** selection disables the display of the Screen Area. You cannot select this screen.
Upper left corner	Defines the upper left corner position of a Screen Area. **Upper left corner**—sets the upper left corner **X** (horizontal) and **Y** (vertical) pixel positions of a Screen Area. • **Signal**—click to select a signal to control the position of the upper-left corner. Because disabled pixels remain visible until overwritten by the pixels in another Screen Area, you need a background Screen Area to see the effects of shrinking a Screen Area. • **Constant**—click to set fixed values the position of the upper-left corner. Dragging a Screen Area in the Layout tab also changes these corner values.
Lower right corner	Defines the lower right corner position of a Screen Area. **Lower right corner**—sets the lower right corner **X** (horizontal) and **Y** (vertical) pixel positions of a Screen Area. • **Signal**—click to select signals to control the position of the lower-right corner. • **Constant**—click to set fixed values for the position of the lower-right corner.

Define Areas Page—About the Enable Property

The **Inspector** tab's **Enable** property selections set the display properties of Screen Areas.

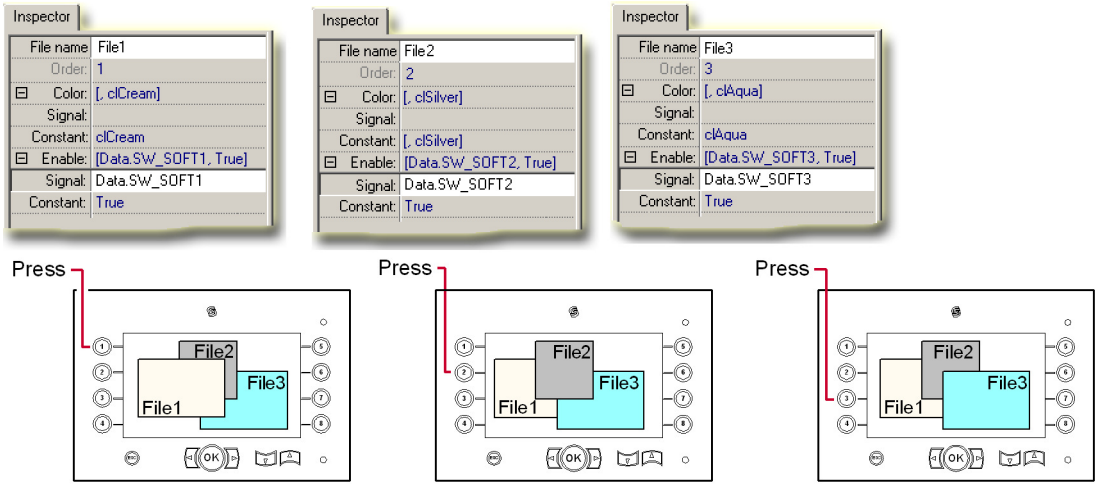
- A signal-enabled Screen Area:
  - Stays invisible after you power on a graphical terminal and becomes visible when its enabling signal becomes True.
  - Moves to the front whenever its enabling signal becomes True.
- A True-enabled Screen Area is always in front until a Screen Area that is signal enabled gets a True signal.

The figure shows how a single True-enabled Screen Area works with two signal-enabled Screen Areas.



Screen Editors

The figure shows how three signal-enabled Screen Areas work together.



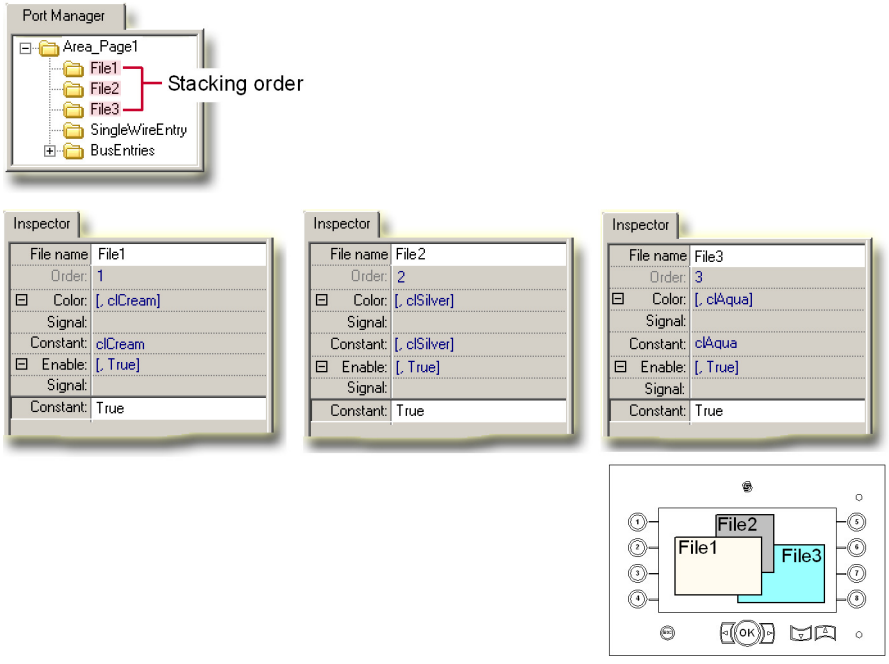
1. Data.SW_SOFT1 = True.
   Data.SW_SOFT2 = False.
   Data.SW_SOFT3 = False.
2. Data.SW_SOFT1 = False.
   Data.SW_SOFT2 = True.
   Data.SW_SOFT3 = False.
3. Data.SW_SOFT1 = False.
   Data.SW_SOFT2 = False.
   Data.SW_SOFT3 = True.

Note the **Enable** settings in the **Inspector** tab for each Screen Area.

Screen Editors

Define Areas Page—About the Order Property

The order in which you assign Screen Areas in the **Port Manager** tab initially sets their front-to-back **Order**.



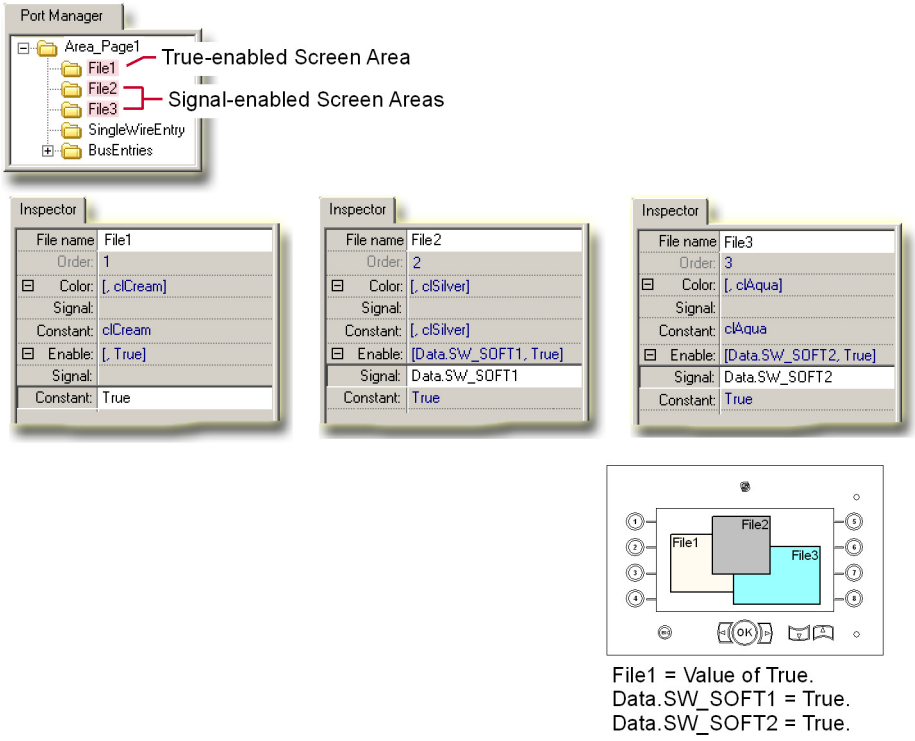
File1, File2, and File3 = Value of True.

The preceding figure shows the display order of three Screen Areas where:

- **File1** Screen Area Enable property is **Constant** with a True value and its **Order** property is **1**.
- **File2** Screen Area Enable property is **Constant** with a True value and its **Order** property is **2**.
- **File3** Screen Area Enable property is **Constant** with a True value and its **Order** property is **3**.

Under these conditions, the **Order** property of a Screen Area sets the front-to-back order in which it displays. To change an **Order** property, right-click a **Screen Area** > **Layout**, in the pop-up menu that opens, click the **Move Forward or Move Backward** command.

Screen Editors



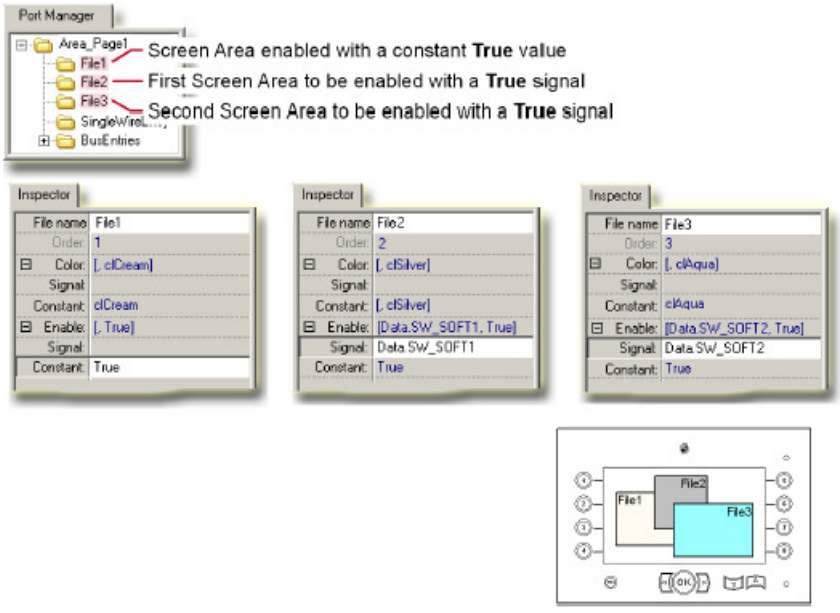
The preceding figure shows the display order of three Screen Areas where:

- The **File1** Screen Area **Enable** property is **Constant** with a True value and its **Order** property is **1**.
- The **File2** Screen Area **Enable** property is **Signal**–enabled by the **Data.SW_SOFT1** signal and its **Order** property is **2**.
- The **File3** Screen Area **Enable** property is **Signal**–enabled by the **Data.SW_SOFT2** signal and its **Order** property is **3**.
- **Data.SW_SOFT1, Data.SW_SOFT2** signals are both simultaneously **True**.

Under these conditions, Screen Areas that are signal-enabled display:

Screen Editors

- In front of any Screen Areas that are enabled with constant trues.
- In a front-to-back order is set by set by their **Order** properties.



The preceding figure shows the display order of three Screen Areas where:

- The **File1** Screen Area **Enable** property is **Constant** with a True value and its **Order** property is **1**.
- The **File2** Screen Area **Enable** property is **Signal**–enabled by the **Data.SW_SOFT1** signal and its **Order** property is **2**.
- The **File3** Screen Area **Enable** property is **Signal**–enabled by the **Data.SW_SOFT2** signal and its **Order** property is **3**.
- **Data.SW_SOFT1** signal first goes **True** and then stays **True**.
- **Data.SW_SOFT2** signal next goes **True** and then stays **True**.

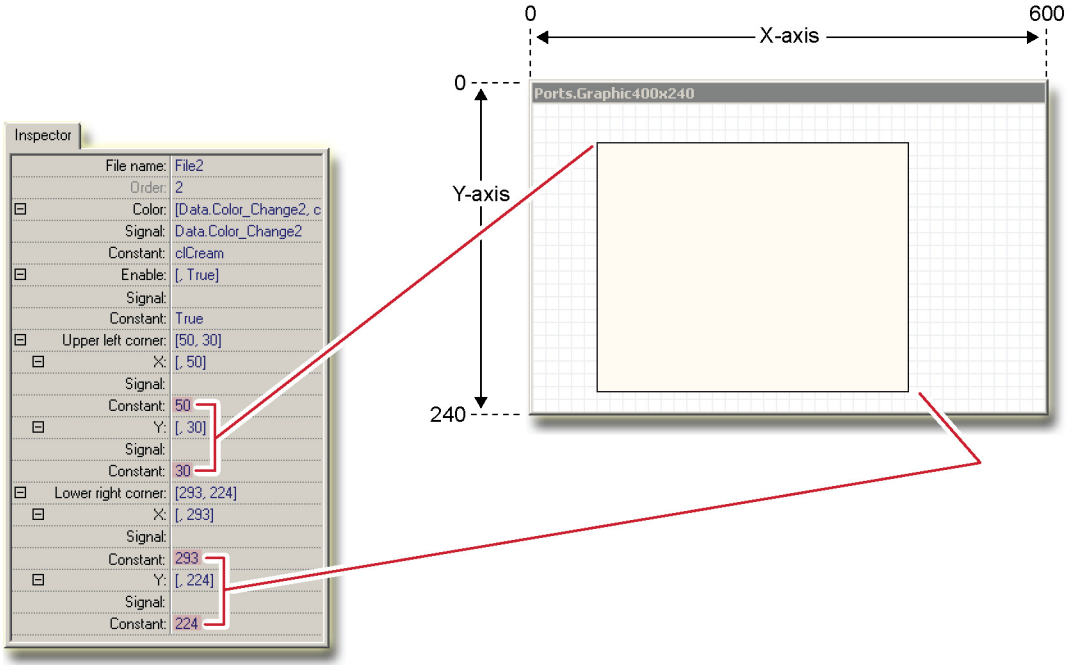
Under these conditions, Screen Areas that are signal-enabled display:

- In front of any Screen Areas that are enabled with constant Trues.
- In a front-to-back order is set by set by the Order in which they are enabled.

Screen Editors

Define Areas Page—About the Corner Property

The preceding figure shows the effect of the **Inspector** tab's **Upper left corner** and **Lower right corner** properties on the size and position of a Screen Area.



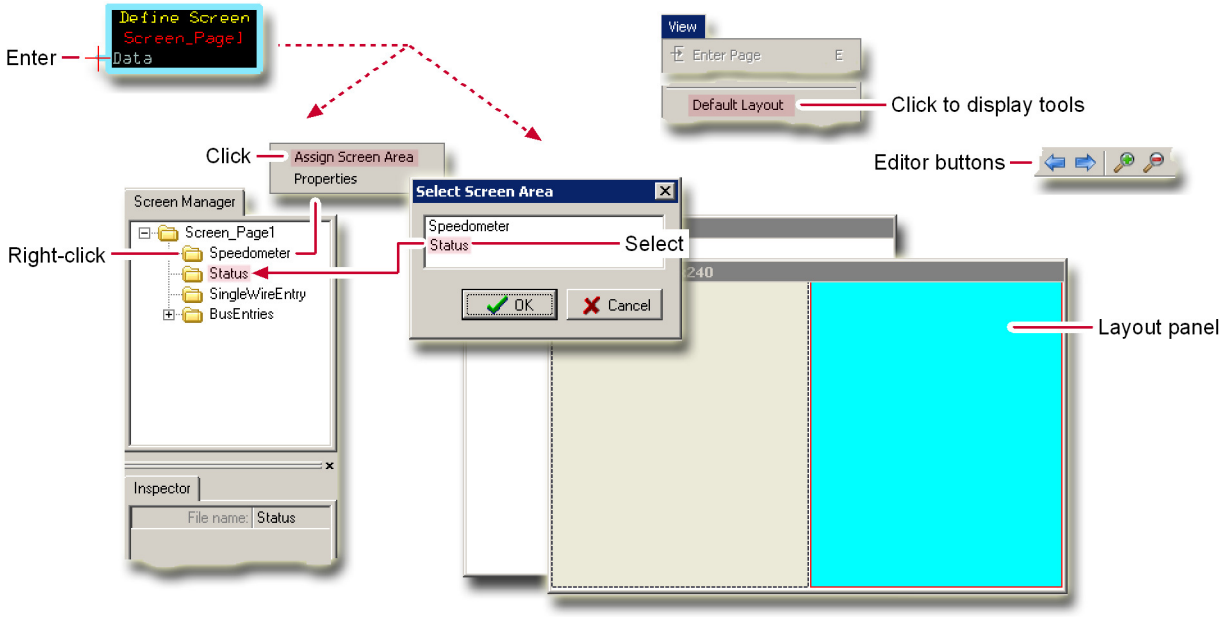
Screen Editors

Define Screen Page

The **Define Screen Page** defines the contents (such as images and text) that appear in Screen Areas. As needed, complete the assignment of the Screen Areas that you defined in the **Define Areas Page**.

The Define Screen Page does not support sub-buses. Always use a main bus to bring signals into this page.

The **File** menu's **Import Page** and **Import Block** commands do not work with the **Define Screens** block. Importing a **Define Screens Page** into a new project strips this page of its contents.



Use the **Screen Manager** tab to manage the assignment of the Screen Areas that you created in the **Define Areas Page**.

- **Screen_Page1**—click for a tree view of the Screen Areas assigned to the **Define Screen Page**. Click a Screen Area name in the tree to bring the area to the front of the Layout tab.
- **SingleWireEntry**—not used here.
- **BusEntries**—click for a tree view of all the signals available through the Data port of the **Define Screen Page**.

Layout tab displays the Screen Areas added to the **Screen1** tree.

(You must return to the **Define Areas Page** if you need to change the property of a Screen Area, such its **Color**, **Enable**, or **Corner** properties.)

Click a Screen Area to bring the area to the front and highlight its name in the **Screen Manager** tab.

The size of your monitor determines the row in which the Editor buttons appear.

Editor buttons/keys are:

 Move backward and forward through your page navigation history.

 , PgDn, PgUp Zoom out and in on the Layout tab.

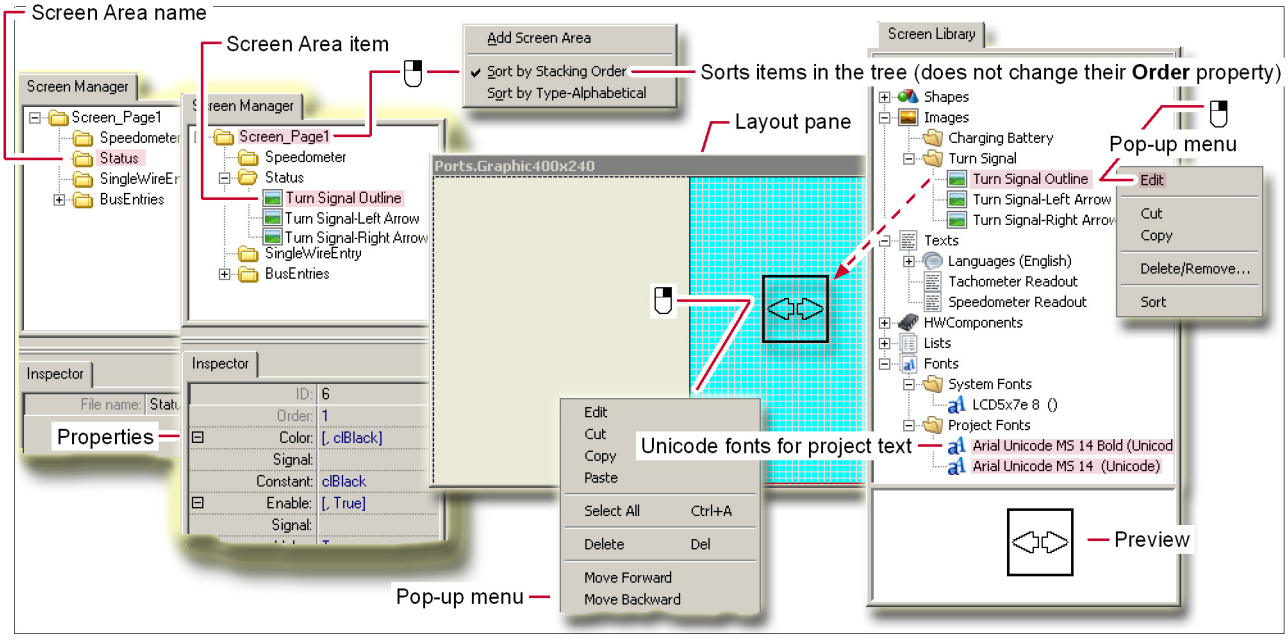
 , L Exit the Classic Screen Editor and return to the Application page.

Screen Editors

Define Screen Page—Add Library Items

Use the tools in the **Define Screen Page** to drag items from the **Screen Library** tab (such as **Shapes**, **Images**, and **Texts**) into Screen Areas and then to manage the properties (such as the fonts applied to text) of these items.

A PLUS+1 display terminal only displays a composite character if it has Unicode support.
(The character Ä, which combines A with ¨, is a composite character.)
A PLUS+1 display terminal cannot display an unsupported composite character, even if the Preview tab in the Classic Screen Editor correctly displays this character.



Add Screen Library Tab Items

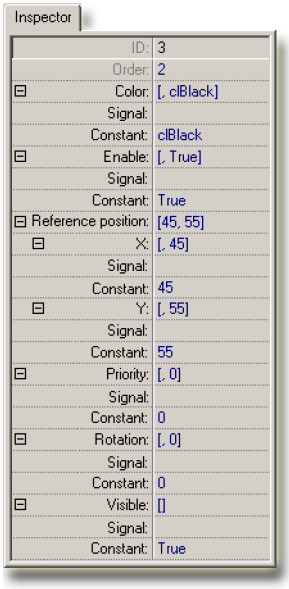
Item	Description
Screen Manager tab	Use this tab to manage the contents of each Screen Area. Click a Screen Area name for a tree view of all the library items that you assign to the selected area. Click Screen Area items in this tab to see their properties in the **Inspector** tab. Right-click items in this tree to open pop-up menus that have **Cut**, **Copy**, **Delete**, and **Paste** commands.
Inspector tab	Use this tab to manage the properties of items placed in Screen Areas.
Layout tab	Drag library items from the **Screen Library** tab into Screen Areas placed here. Click a Screen Area to bring the area to the front and highlight its name in the **Screen Manager** tab. Click a library item to see its properties in the **Inspector** tab. The **Screen Manager** tab highlights the name of the selected item. Right-click items in the Layout pane to display a pop-up menu. In this pop-up menu, use the: • **Edit** command to display the selected item in the Image Register. • **Move Forward**, **Move Backward**, **Bring to Front**, and **Send to Back** commands to change an item's **Order** property.
Screen Library tab	Has items such as **Shapes**, **Images**, **Texts**, and Unicode fonts that are available for use in your graphic project. • Use the **Image Register** tab to add to the library of available **Images**. • Use the **Text Register** tab to add to the library of available **Texts**.
Preview tab	Displays a preview of items that you select in the **Screen Library** tab.

Screen Editors

Define Screen Page—Inspector Tab

Use this tab to manage the properties of the library items that you add to Screen Areas. To view the properties of a library item, click the item name in the **Screen Manager** tab or click the item itself in the Layout tab. Different types of library items have different properties, so that the type of item you select determines the properties displayed by the **Inspector** tab.

See *Define Screen Page/About Screen Item Properties* on page 442 for a list of the properties of **Shapes**, **Images**, **Texts**, and **HWComponents**.



Inspector Tab

Item	Description
ID	Used for internal identification purposes. You cannot change this value.
Order	Indicates the front-to-back display order of overlapping library items when library items simultaneously have an **Enable** with a True **Signal** or a **Constant** that is **True**. An item with an **Order** of **1** displays in front of a library item with an **Order** of **2**. The sequence in which you drag a library item into the Layout pane sets its initial **Order**. To change a library item's **Order**, right-click the item. In the pop-up menu that displays, click the **Move Forward**, **Move Backward**, **Bring to Front**, or **Send to Back** command.
Color	Sets the color property of the library item. Click **Signal** to select a signal from a list of signals to set the color. Click **Constant** to select a color from a list of colors. Double-click **Constant** to select a color from a standard Windows color palette.
Enable	Enables the display of library items.
Reference position	Sets the **X**-axis and **Y**-axis coordinates of the small blue cross displayed on **Images** and **HWComponents** library items. All vertical, horizontal, and rotational movements of the library item take place with reference to this point. **Reference point** in the Image Register tab sets the position of this cross.
Begin point	**X**—sets the starting y-axis (horizontal) position, in pixels, of a **Line**. **Y**—sets the starting y-axis (vertical) position, in pixels, of a **Line**. (You select **Line** from the **Shapes** tree in the **Screen Library** tab.)
End point	**X**—sets the ending x-axis (horizontal) position, in pixels, of a **Line**. **Y**—sets the ending y-axis (vertical) position, in pixels, of a **Line**.
Width	Sets the top-to-bottom thickness, in pixels, of a Line drawn on the horizontal x-axis. Sets the side-to-side thickness, in pixels, of a Line drawn on the vertical y-axis.
Font	Sets the font property of **Texts** taken from the **Screen Library**.

Screen Editors

Inspector Tab (continued)

Item	Description
Priority	Determines which library item displays when two or more items simultaneously have an **Enable** with a True **Signal**. When you enable two or more items, the graphic terminal only displays the item (or items) with the highest **Priority**. Lower **Priority** numbers give library items higher display priorities, with **0** giving an item the highest display priority.
Rotation	Rotates **Images** and **HWComponents** library items. All rotational movements take place with reference to the small blue cross that displays on these library items. **Reference point** in the Image Register pane sets the position of this cross. Apply rotational values between **0** and **359°**.
Visible	Outputs a True Boolean signal when a library item becomes visible. To be visible, a library item must have a True Screen Area, a True **Enable** property, and a **Priority** that is higher than or equal to the **Priority** of any other library item in the same Screen Area. (You must connect a wire to the **Data** bus to be able to output this signal.)

Define Screen Page/About Screen Item Properties

Different types of library items have different properties. This table lists the properties of **Shapes**, **Images**, **Texts**, and **HWComponents**.

Properties of Screen Library Tab Items

Property	Library Items			
	Shapes	**Images**	**Text**	**HW Components**
Begin point	yes	—	—	—
Color	yes	yes	yes	yes
Enable	yes	yes	yes	yes
End point	yes	—	—	—
Font	—	—	yes	—
ID	—	yes	yes	yes
Reference position	—	yes	—	yes
Order	yes	yes	yes	yes
Priority	yes	—	yes	yes
Rotation	—	yes	—	yes
Starting point	—	—	yes	—
Visible	yes	yes	yes	yes
Width	yes	—	—	—

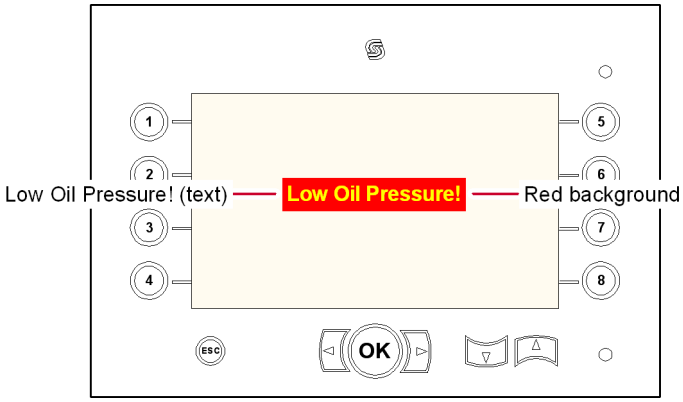
Screen Editors

Define Screen Page/About the Order Property

The **Inspector** tab has **Order** and **Enable** properties that work together to produce different display results.

The **Inspector** tab shows the stacking order of display items. In the following example the:

- **Low Oil Pressure!** text has an **Order** of **1**, with **Enable** properties that change.
- Red background has an **Order** of **2**, with **Enable** properties that change.



Order and Enable Properties

Item	Order	Enable	Result
Low Oil Pressure!	1	Constant	Constant display, with alert text in front of background bar.
Red background	2	Constant	
Low Oil Pressure!	1	Signal	Constant display of background bar. Alert text only displays in front of background bar when its signal becomes True.
Red background	2	Constant	
Low Oil Pressure!	1	Signal	Alert text and background bar only display when their signals become True. Alert text displays in front of background bar when both signals are True.
Red background	2	Signal	
Low Oil Pressure!	1	Constant	Constant display of alert text. Background bar displays behind alert text when its signal becomes True.
Red background	2	Signal	

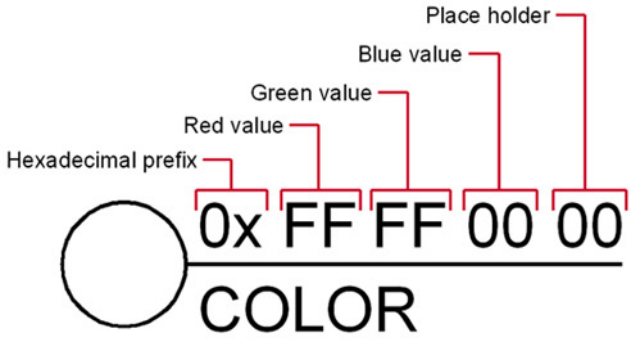
Screen Editors

Define Screen Page/About the Color Property

The **Inspector** tab has a **Color** property that allows you to set the color through a **Constant** or through a **Signal**.

When setting the color through a **Signal**, use a **Multi-character Constant** component to output red, green, and blue (RGB) color values.

Use the ten-character scheme shown below when setting individual RGB values.



Two-character hexadecimal values set the red, green, and blue values of the color where:

- A hexadecimal value of FF equals a decimal value of 255.
- A hexadecimal value of 00 equals a decimal value of 0.

Color Examples

Constant	Red Value		Green Value		Blue Value		Color
	Hex	Decimal	Hex	Decimal	Hex	Decimal	
0x FF 00 00 00 COLOR	FF	255	00	00	00	00	Red
0x FF 00 00 00 COLOR	00	00	FF	255	00	00	Green
0x FF 00 00 00 COLOR	00	00	00	00	FF	255	Blue
0x FF 00 00 00 COLOR	FF	255	FF	255	00	00	Yellow

Screen Editors

Define Screen Page/About the Priority Property

The **Inspector** tab has a **Priority** property that determines which library item displays when two or more items simultaneously have an **Enable** with a True **Signal**.

When you enable two or more items, the display terminal only shows the item (or items) with the higher **Priority**.

- An item with a **Priority** of **0** displays before an item with a **Priority** of **1**.
- An item with a **Priority** of **1** displays before an item with a **Priority** of **2**.

In the following example:

- The **Low Oil Pressure**, **Dirty Air Filter**, and **Low Washer Fluid** alerts all appear within the same Screen Area.
- The alerts in order of importance are:
  - **Low Oil Pressure**.
  - **Dirty Air Filter**.
  - **Low Washer Fluid**.
- Only the three alerts appear in this Screen Area.
- Each alert is a bitmap, composed of yellow text against a red background.

Priority example

Alert	Enable Signal Status	Priority	Display
Low Oil Pressure	True	0	Low Oil Pressure
Dirty Air Filter	True	1	
Low Washer Fluid	True	2	
Low Oil Pressure	False	0	Dirty Air Filter
Dirty Air Filter	True	1	
Low Washer Fluid	True	2	
Low Oil Pressure	False	0	Low Washer Fluid
Dirty Air Filter	False	1	
Low Washer Fluid	True	2	

Screen Editors

Define Screen Page—Image Register

Use the Image Register tab to make BMP images available for use in graphical terminal projects. Registering images:

- converts JPEG, TIFF, and GIF image files to BMP image files. (The original JPEG, TIFF, and GIF image files do not change.)
- copies image files that are outside of the project folder into the project folder.
- determines how a graphical terminal displays images by setting properties such as **Color depth**.
- makes images available for use in the project through the **Screen Library** tab's **Images** tree.

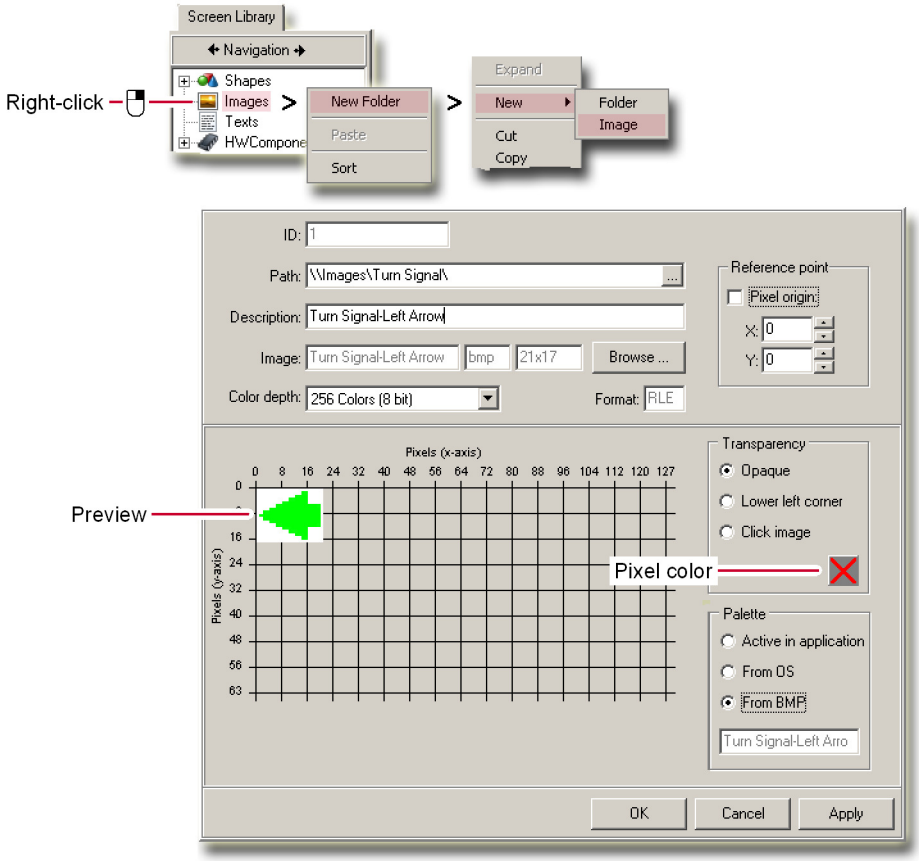


Image Register tab

Item	Description
ID	Used for identification purposes. You cannot change this value.
Path	Displays the location of the registered image in the **Screen Library** tab's **Images** tree.
Description	Enter the reference name for the bitmap that you are registering. This name displays in the **Images** tree when you click **Save**.
Image	Displays the name of the source file, its image type, and its size in pixels.
Browse	Click to display an **Open** window. Locate and select the source bitmap file in this window. The Classic Screen Editor copies the file that you select into the graphic project folder.
Reference point	Sets an **X**-axis and **Y**-axis reference point on bitmap images. All vertical, horizontal, and rotational movements of the bitmap image take place with reference to this point.
Pixel origin	Becomes available if your image has a single pixel with a unique color. Check to set the **Reference point** on this pixel.
Color depth	Selects the color depth that the bitmap image displays.

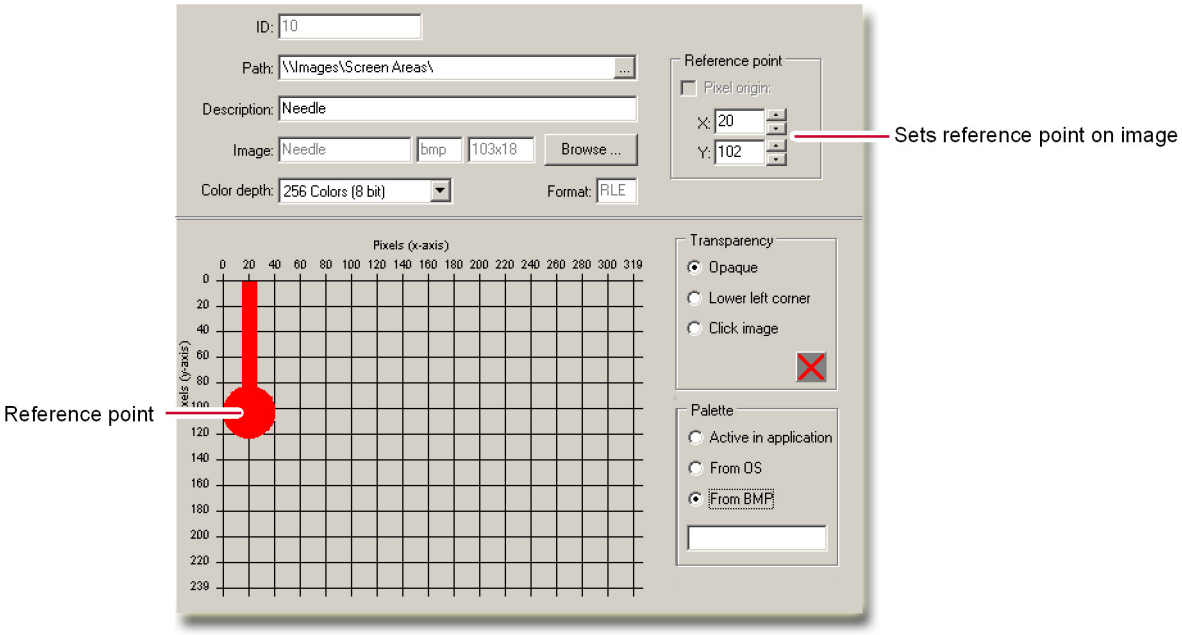
Screen Editors

Image Register tab (continued)

Item	Description
Format	Displays the compression technique used on the bitmap image.
Preview	Displays a preview of the bitmap image.
Transparency	Click to set what areas of the bitmap image are transparent. **Opaque**—all pixels in the bitmap image are visible. **Lower left corner**—all pixels with the same color as the pixel in the lower left corner of the bitmap image become transparent. **Click image**—all pixels in the bitmap image that have the same color as the pixel that you click become transparent. **Pixel color**—displays the color of the selected pixel.
Palette	Sets the color palette that the bitmap image uses. **Active in application**—uses a color palette previously defined in the application. **From OS**—uses the color palette defined in the hardware operating system. **From BMP**—uses the color palette from the BMP source file.
OK	Registers the bitmap image and closes the Image Register tab. The bitmap source file copies to the project folder. The **Image** tree in the **Screen Library**tab displays the name of the registered image.
Cancel	Closes the Image Register pane without registering any images.
Apply	Registers the bitmap image and leaves the Image Register tab open. The bitmap source file copies to the project folder. The **Image** tree in the **Screen Library**tab displays the name of the registered image.

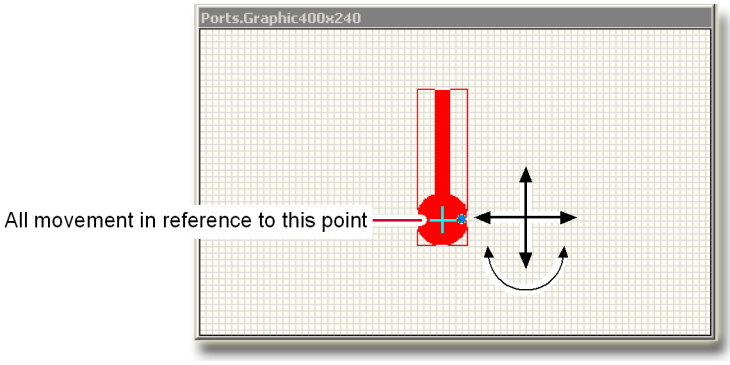
Screen Editors

Define Screen Page/About the Reference Point



The **Reference point** sets an **X**-axis and **Y**-axis reference point on an image.

The default **Reference point** is the upper left corner of the image.



When you click an image in the Layout pane, a small blue cross identifies the **Reference point**.

All vertical, horizontal, and rotational movements of the image take place with reference to this point.

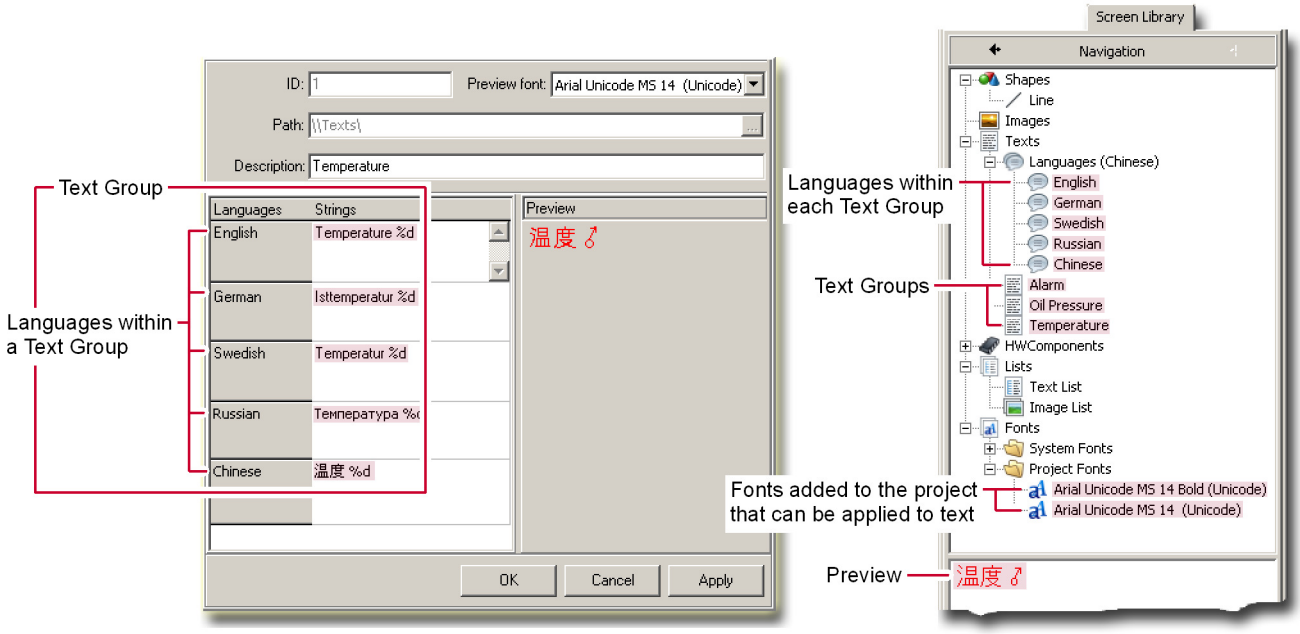
Screen Editors

Define Screen Page—Text Register

Use the **Text Register** tab to add Text Groups to the **Texts** tree in the **Screen Library** tab.

Text groups organize the texts displayed by a graphical terminal. Registering a **Text Group**:

- Copies the strings in the **Text Group** to a file in the graphic project folder.
- Defines (registers) the display characteristics of each text string.
- Displays the **Description** name of the **Text Group** in the **Screen Library** tab's **Texts** tree.



Text Register tab

Item	Description
ID	Used for identification purposes. You cannot change this value.
Preview font	Sets the font and size of the string that displays in the **Preview** pane. (The **Inspector** tab's **Font** property sets the font that a graphical terminal uses when it displays the string.)
Path	Displays the location of **Text Group** within the **Texts** tree in the **Screen Library** tab.
Description	The name for the **Text Group** that appears in the **Screen Library** tab's **Texts** tree when you click **Save**. Click a **Text Group** name in the **Texts** tree to open the Text Register pane and view the strings in the group.
Languages	Lists the different languages in which you can display the **Text Group** string.
Strings	Enter the text that you want to display here. • use **CTRL + ENTER** for line breaks. • use the **%** print character to display data values in the string.
Preview tab	Provides a preview of your **String** entry. (The **Inspector** tab's **Font** property sets the font that a graphical terminal uses when it displays the string.)
OK	Registers the **Text Group** and closes the Text Register tab. The strings in the group copy to a text file in the graphic project folder. The **Texts** tree in the **Screen Library** tab displays the **Description** of the **Text Group**.
Cancel	Closes the Text Register pane without making any **Text Group**changes.
Apply	Registers the **Text Group** and leaves the Text Register tab open. The strings in the group copy to a text file in the graphic project folder. The **Texts** tree in the **Screen Library** tab displays the **Description** of the **Text Group**.

Screen Editors

Vector-Based Screen Editor

Your graphical hardware determines your screen editor choice. The PLUS+1 GUIDE application has these two screen editors:

- Vector-Based Screen Editor, which you use with the **Show Screen** component and **Screen Definitions** to create an application for a PLUS+1® graphical terminal.

  The **Show Screen** component and **Screen Definitions** are hardware-dependent components. They become available after you install a hardware file (HWD) in the **Project Manager** tab that supports the graphical terminal for which you are creating an application.
- Classic Screen Editor, which you use with the **Define Areas Page** and the **Define Screen Pages** components to create an application for a PLUS+1® graphical terminal.

  For some basic "how-tos" about the Classic Screen Editor, refer to *Classic Screen Editor* on page 428.

Screen Editors

Elements of the Vector-Based Screen Editor

- Screen assets define the texts and images that appear in your application's screens.
- **Screen Definitions** define the screens that appear in your application.
- **Screen Definitions** contain screen assets.
- The same screen assets are available for use in all **Screen Definitions**.
- The same screen asset can be "instantiated" (used) in many different **Screen Definitions**.
- The same screen asset can have different properties in different **Screen Definitions**.
- The same **Screen Definition** can be "instantiated" (used) in different places in your project.
- The **Show Screen** component instantiates **Screen Definitions**.
- You place **Show Screen** components in your application to instantiate **Screen Definitions**.
- The **Show Screen** component connects the signals between the application and the **Screen Definition** that it instantiates.

Danfoss Recommends the SVG Format

The Vector-Based Screen Editor can rescale objects. Rescaling simplifies the task of adapting an application for use on terminals with different-sized screens.

The Vector-Based Screen Editor supports the SVG (Scalable Vector Graphic) format. This format provides the smoothest rescaling of graphical objects.

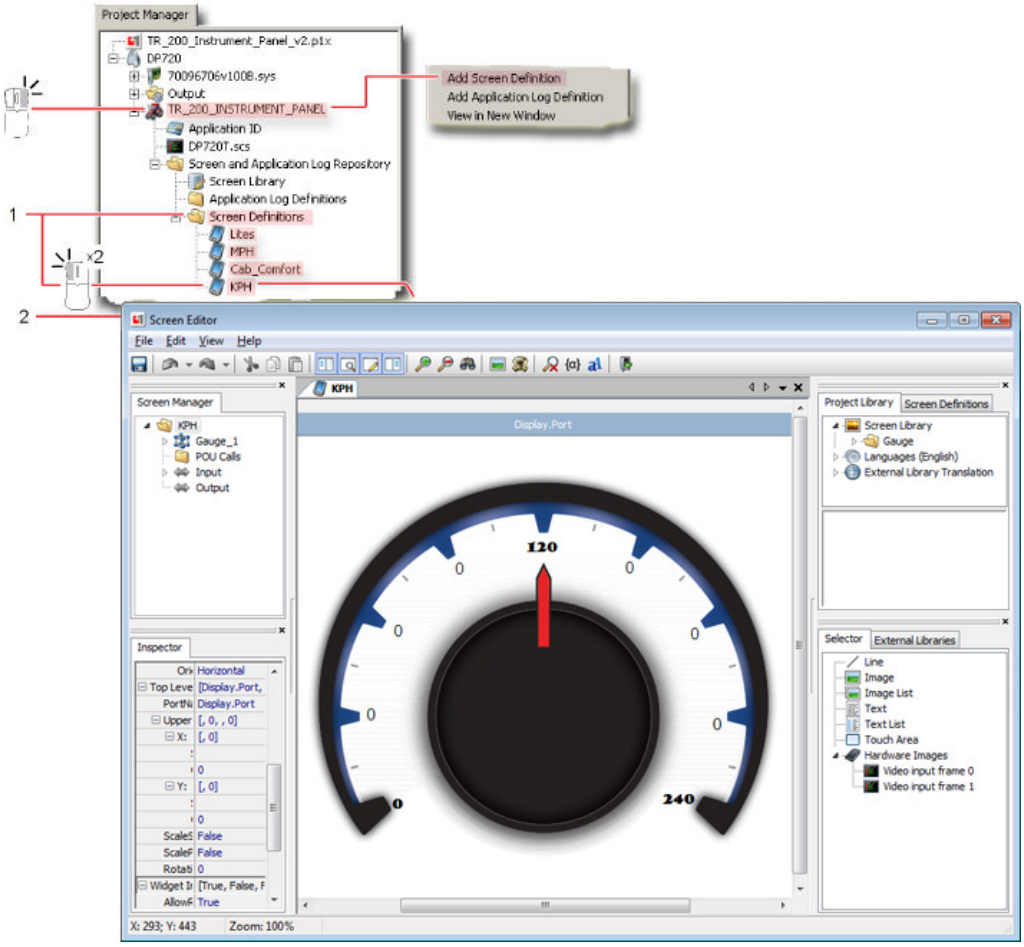
The Vector-Based Screen Editor also supports the JPEG, GIF, BMP, PNG, and TIFF formats. However, rescaling objects in these formats can produce a jagged and distorted output.

Danfoss recommends that you use the SVG format whenever possible.

Screen Editors

About Screen Definitions and the Screen Editor Window

Use the Screen Editor window to create Screen Definitions



1. **Screen Definitions** define the screens in a PLUS+1 GUIDE graphical application. The **Project Manager** tab lists all the Screen Definitions that are available in a project. Right-clicks in this tab open a pop-up menu with **Copy**, **Add**, **Rename**, **Delete**, and **Add Screen Definition** commands. The **Project Manager** tab in the preceding figure has Screen Definitions for:
   - **Lites**
   - **MPH**
   - **Cab_Comfort**
   - **KPH**
2. **Screen Editor Window** creates the contents of a **Screen Definition** in the **Screen Editor** window, mostly from screen assets listed in the **Screen Library** tab. These screen assets are available for instancing (use) in every **Screen Definition** that you create.

   Double-click a **Screen Definition** to open the selected **Screen Definition** in the **Screen Editor** window.

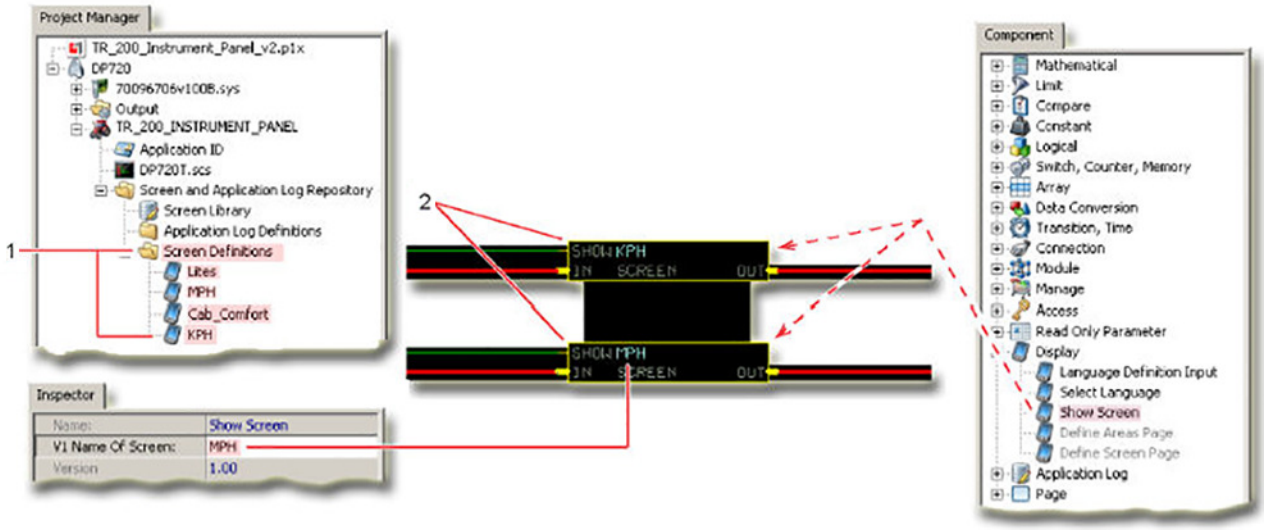
Screen Editors

About Show Screen Components and Screen Definitions

Show Screen components instantiate **Screen Definitions**.

The signals routed to a **Show Screen** component's **IN** and **OUT** buses connect the application and the **Screen Definition** instantiated by the **Show Screen** component.

- You must type all the signals that you connect to a **Show Screen** component's **IN** bus.
- You can connect sub-buses to a **Show Screen** component's **IN** and **OUT** buses.



1. **Screen Definitions** — The **Project Manager** tab lists all the **Screen Definitions** that are available in your project. The project in the preceding figure has four available **Screen Definitions**:
   - **Lites**
   - **MPH**
   - **Cab_Comfort**
   - **KPH**
2. **Show Screen** components instantiate **Screen Definitions**.
3. To instantiate a Screen Definition, query the Show Screen component. Then enter the name of the Screen Definition that you want to instantiate in the Inspector tab's V1 Name of Screen field.

   Each **Screen Definition** requires an instancing **Show Screen** component to display.

   Show Screen components can call the same **Screen Definition** wherever it is needed in a graphical application.

   The project in the preceding figure has two **Show Screen** components:
   - The **Show Screen** component with the **ScreenId** of **KPH** instantiates the **KPH Screen Definition**.
   - The **Show Screen** component with the **ScreenId** of **MPH** instantiates the **MPH Screen Definition**.

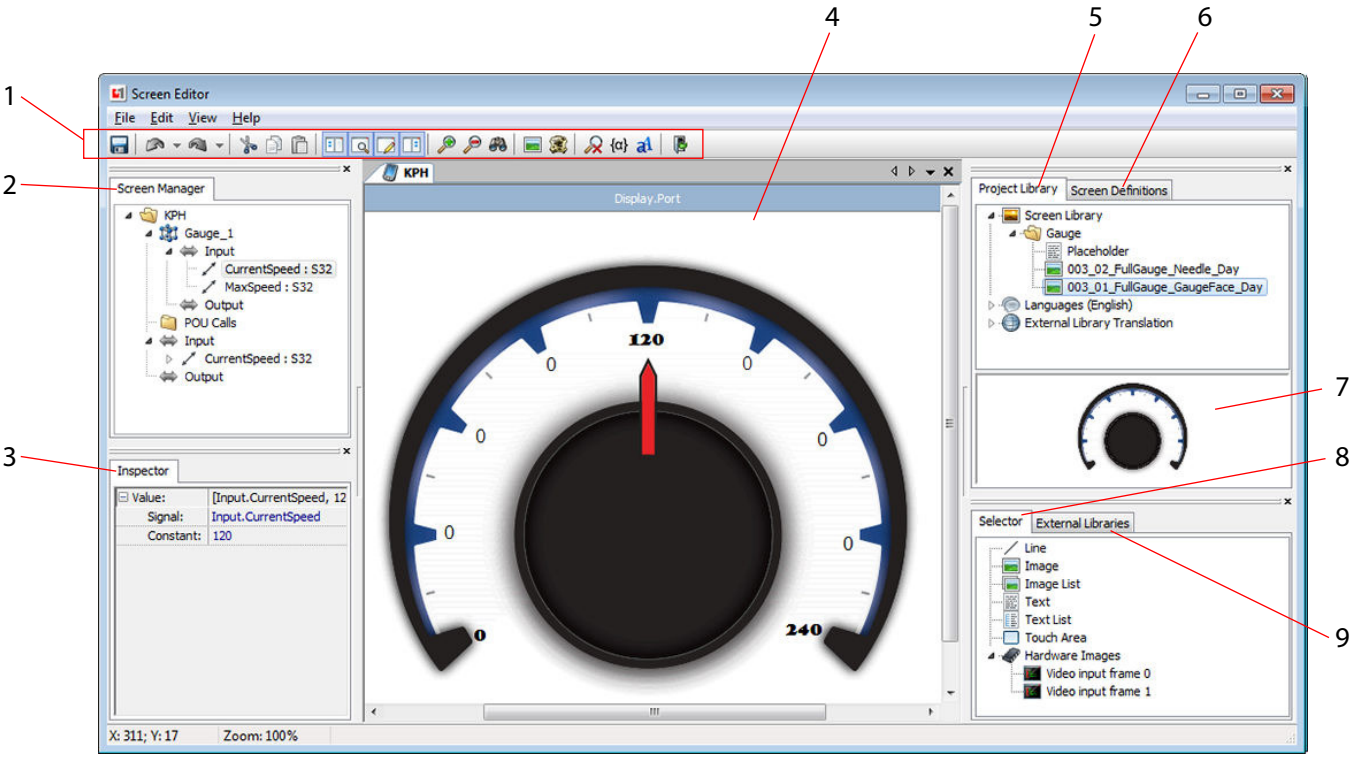
     Signals routed to these two **Show Screen** components control the display of each **Screen Definition**.

     The **Screen Definitions** for **Lites** and **Cab_Comfort** have no instancing **Show Screen** components. The application will not display these two **Screen Definitions**.

Screen Editors

Screen Editor Window

Use the Screen Editor window to create Screen Definitions



Screen Editor Window

Item	Name	Description
1.	**Toolbar**	Click these buttons for common functions such as zoom, undo, redo, hide/display grid, and exit.
2.	**Screen Manager** tab	Use this tab to manage the screen assets displayed in the Design Area. The Inspector tab displays the properties of items that you click in the **Screen Manager** tab.
3.	**Inspector** tab	Use this tab to review and assign properties to screen assets that you click in either the **Screen Manager** tab or in the Design Area. The type of item that you click in the Screen Manager tab or Design Area determines the type of properties that the **Inspector** tab displays.
4.	**Design Area**	Design the appearance of screens here. Double-click to select an item and open its **Common Properties** window.
5.	**Project Library** tab	Lists and organizes the screen assets—such as images and texts—that are available for use in your project. Every **Screen Definition** that you create has access to the same screen assets.
6.	**Screen Definitions**	Displays a tab that mirrors the **Screen Definition** node shown in the **PLUS+1 GUIDE** window's **Project Manager** tab. Click on the **Screen Definition** shown in this tab to view and edit other Screen Definitions in your project. Drag a screen definition to the Design Area to add it as a widget.
7.	**Preview** tab	Provides thumbnails of the screen assets that you click in the **Screen Library** tab.
8.	**Selector** tab	Drag items from this tab into the Design Area. Then edit these items to create screen assets. All screen assets that you create (except lines) become available through the **Screen Library**.
9.	**External Libraries** tab	Drag widgets from libraries to the Design area

Project Library Tab

The Screen Library contains all screen assets added to a PLUS+1 GUIDE project. Assets can be organized in a folder structure.

The languages of the project are also managed in the Screen Library. For each language, there is one translation in all text definitions. In run-time, the translation that belongs to the currently active language

Screen Editors

is displayed for all text objects. See *Allow a User to Change Languages* for information on how to set the currently active language in run-time.

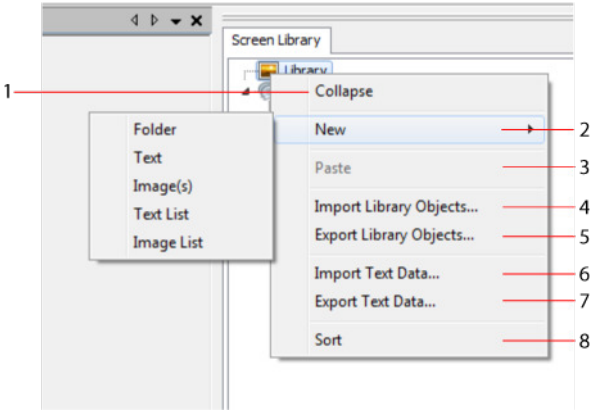
Use **External Library Translation** to map translations and add missing translations for library widgets.

Library

Available assets

Name	Description
Text Definition	A text that has as many translations as there are languages in the application and up to 20 data values.
Image	A referenced physical image file and a set of attributes applied to that image. If the application has a boot logo image (HWD dependent), it will be available under a folder called `Boot logo`. Editing a boot logo image in the **Edit Image** window will affect its appearance in screen definitions, but not when the display is booting. It is not possible to delete a boot logo image or replace its source file in Vector-Based Screen Editor.
Text List	An ordered list of text definitions. An index input determines which of the texts to show.
Image List	An ordered list of image definitions. An index input determines which of the images to show.

Context menu used to manage the screen library



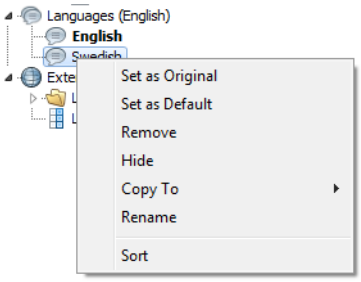
Item	Name	Description
1.	Collapse/Expand	Collapse or expand the **Screen Library** tree view. If the tree view is expanded, Collapse will be available and the other way around.
2.	New	Add a new folder or screen asset. Folder – add a folder. Text – add a new text definition. The Edit Text dialog will be opened. Image – add new image definitions. A file browser will be opened where it will be possible to select one image, multiple images, of a folder structure containing images. The selected images will be copied to the project folder, and then the **Edit Image** dialog will be opened. Text List – add a new text list. Image List – add a new image list.
3.	Paste	Paste the currently copied screen library items or folder structure. Enabled if screen library items have been copied to the clipboard.
4.	Import Library Objects	Import a screen library sub-structure from file. See *About Exporting and Importing Library Objects* on page 522.
5.	Export Library Objects	Import a screen library sub-structure from file. See *About Exporting and Importing Library Objects* on page 522.
6.	Import Text Data	Import text definitions. This command may be used for importing texts from a PLUS+1® GUIDE project that uses Classic Screen Editor.

Screen Editors

Item	Name	Description
7.	Export Text Data	Export all text definitions. Note, **Export Library Objects** should be the preferred way to export text definitions.
8.	Sort	If checked, all items in the **Screen Library** will be sorted alphabetically.

Languages

Right-click **Languages** to add a new language to the application. Languages are managed through the **Context** menu.



Item	Description
Set as Original	Make this language the **Original Language.**a PLUS+1® GUIDE application has an **Original Language** and a number of translations.
Set as Default	Make this language the **Default Language.** The **Default Language** is shown for all text objects in the **Design Area.**
Remove	Remove the language and all translations of that language from the application.
Hide	Hide/show a language.
Copy To	Copy all texts written in this language to the target language. This will overwrite any texts written in the target language.
Rename	Rename this language.
Sort	If checked, all languages will be sorted alphabetically.

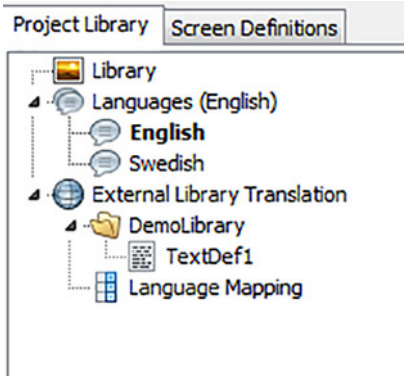
External Library Translation

Manage translations of texts in library widgets. **External Library Translation** has one sub-node for each library from which widgets are used in the application. Such a sub-node contains all text definitions used in any library widget added to the project. In case the widget library has no translations for a language used in the application, translations can be added by editing the text definitions in the library node.

When a library widget has been removed, resulting in a text definition becoming unused, that text definition will be greyed out. Use the context menu of a library sub-node to delete all unused translations.

Use the **Language Mapping** node to map languages between the library and the application. Existing languages with an exact match will be mapped automatically.

Screen Editors



External Library Tab

The **External Library** tab contains all libraries installed in PLUS+1® GUIDE see *Function* on page 128. Widgets in libraries (library widgets) can be dragged to a screen definition. The definition of a library widget cannot be opened or edited. To compile a project containing library widgets, the corresponding libraries must be installed. Adding a library widget to an application will not add any images or POUs used by that library widget. Library widgets have three searchable properties; **WidgetLibrary**, **WidgetId** and **WidgetVersion**.

Library widgets may be extendable. An extendable library widget can be imported to an application using the context menu. Once imported, any properties or constraints related to library widgets are removed and it becomes available in the **Screen Definitions** tab.

Screen Editors

Edit Image Window

Use the **Edit Image** window to:

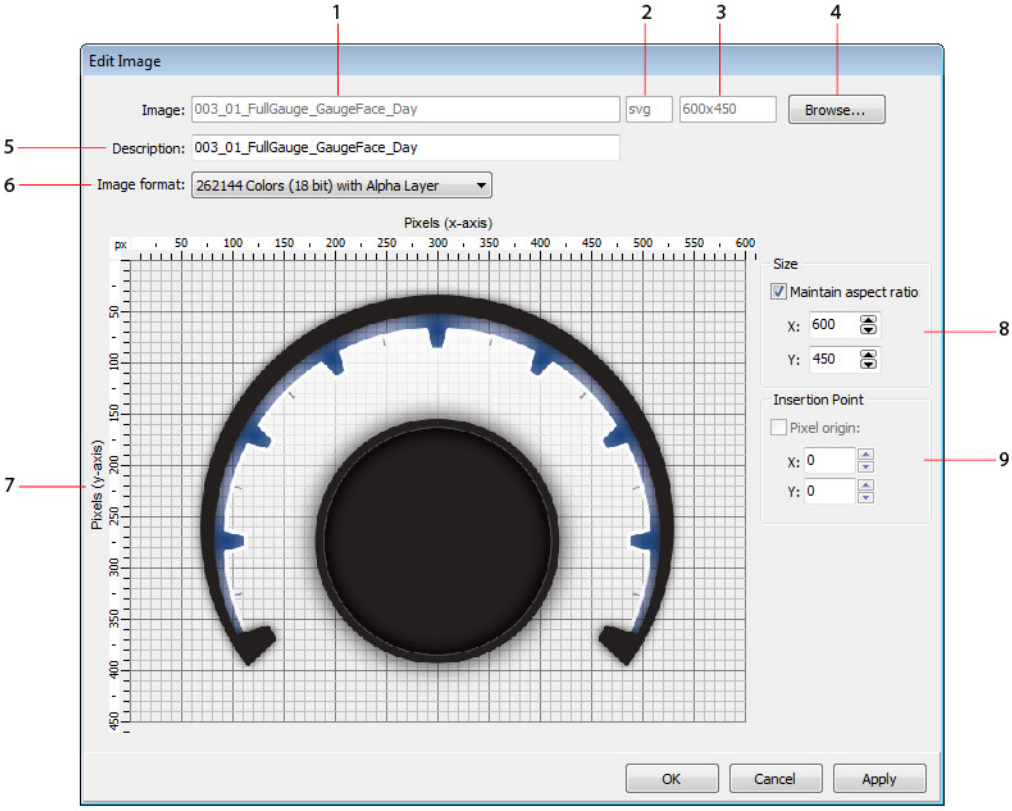
- create an image asset and define its basic graphical properties.
- make the image asset available for instancing (use) in your project's **Screen Definitions**.
- edit the graphical properties of an image asset that is already in your project.

The **Library** tab lists all the image assets that you add to your project. These image assets are available to every **Screen Definition** that you create in your project.

Drag the asset from the **Library** tab into the Design Area to instantiate (use) an image asset in a **Screen Definition**.

Use the **Common Properties** window or the **Inspector** tab to define the control signals, position, and other properties of the image assets instantiated in the Design Area. The same image asset can have different properties in different **Screen Definitions**. See *Common Properties Windows* on page 466 and *Inspector Tab* on page 474.

The **Image Editor** window has a **Click image** option for pixel **Transparency**. Rescaling an image with pixels made transparent with this option causes a coordinate shift in the location of the pixel selected for transparency. In some cases, where you have different colored pixels next to the selected pixel, your rescaled image may have different transparent pixels. Danfoss recommends that you use alpha-channel images and that you set transparencies in these images before importing them into the Screen Editor.



Edit Image Window

Item	Name	Description
1.	Image file name	The name without extension of the image file.
2.	File extension	File extension of the image.
3.	Original image size	Image size.

Screen Editors

Edit Image Window (continued)

Item	Name	Description
4.	Browse button	Open a file dialog to select another physical image file.
5.	Description	A descriptive name of the image. By default, this will be the image file name without extension.
6.	Image format	File format that will be used in the PLUS+1® GUIDE application. This is determined automatically when an image is being added. Selected image format will have an impact on the properties available for the image definition. See *Image Formats* on page 459.
7.	Image preview	A preview of the image when the selected Image format is applied. The preview is scaled to fit the dialog.
8.	Image definition size	Set the size of the image definition to control the default size of its instances in the design area. This property is only available for vector-based image formats (such as, SVG).
9.	Insertion point	An offset to the insertion point. When an image instance is added, its insertion point will match its (X, Y) coordinates. The image instance will rotate around this point if it is rotation. By default this value is (0, 0), corresponding to the top left corner of the image instance.

The maximum size of an image instance may be limited by the HWD. When an image instance that is too large is added or an attempt is made at resizing an image instance beyond the limits, the size of the image instance will be capped to the maximum size defined by the HWD and a warning message will be shown.

Image Formats

The image format setting of an image definition specifies the appearance of the image in the target display and may add more configuration options in Vector-Based Screen Editor. Available image formats are defined in the HWD file.

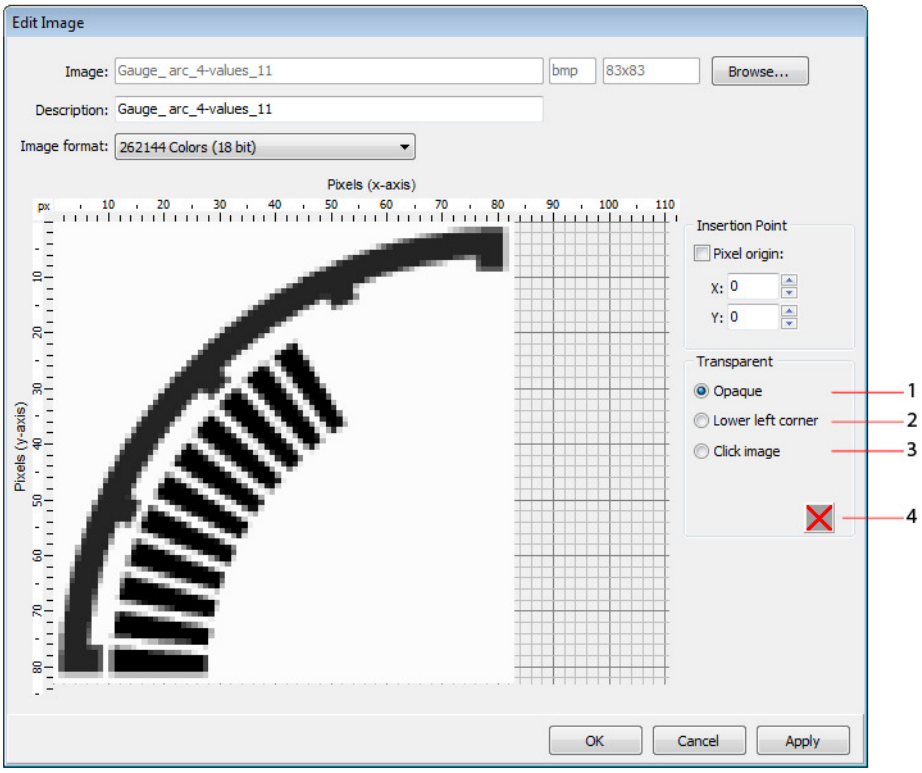
The three kinds of image formats

Name	Description
Color/grayscale formats with alpha channel	Images of this format contain an alpha channel, which allows each pixel to have a transparency between fully transparent and fully opaque.
Color/grayscale formats without alpha channel	Images without alpha channel have an option to specify a color that will be fully transparent on the target display.
Monochrome (1 bit) image format	Each pixel in a monochrome image is either colored or transparent. Which color to use is specified for each instance of the monochrome image.

Selecting transparency

In the **Edit Image** dialog, a panel containing transparency settings will appear when an image format without alpha channel is selected.

Screen Editors



Item	Name	Description
1.	Opaque	If selected, there will be no transparent pixels in the image.
2.	Lower left corner	All pixels sharing the color of the pixel in the lower left corner will be transparent.
3.	Click image	When selected, the transparent color is selected by clicking it in the preview.
4.	Transparent color	Preview of transparent color. If Opaque is selected, a red cross will appear.

Screen Editors

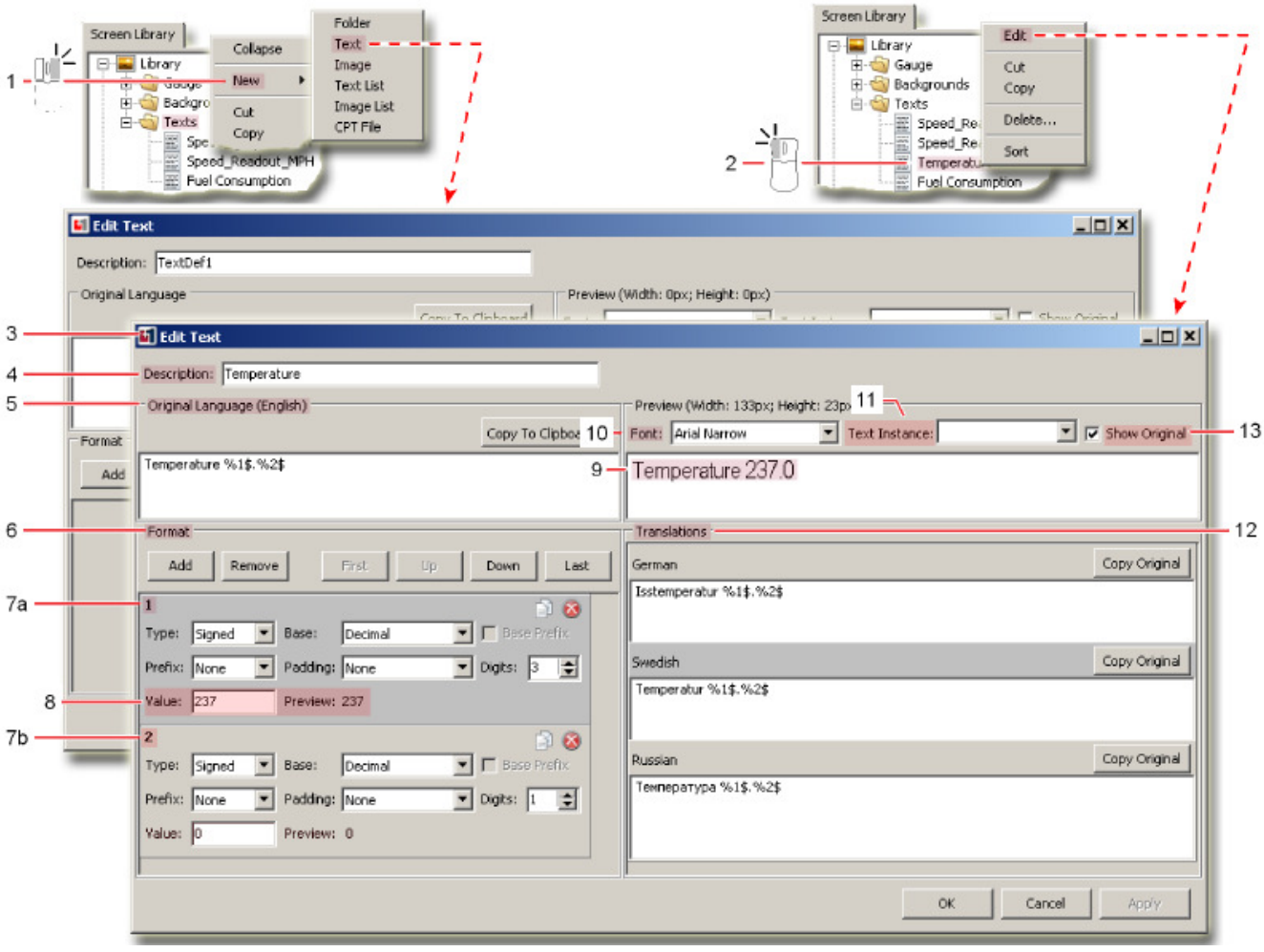
Edit Text Window

Use the **Edit Text** window create or edit a text asset.

The **Library** tab lists all the text assets in your project. These text assets become available for instancing in every **Screen Definition** that you create for your project.

To instantiate (use) a text asset in a **Screen Definition**, drag the text asset from the **Library** tab into the Design Area.

The **Edit Text** window in the following figure creates a Temperature text asset that reads out engine coolant temperatures.



Edit Text window

Item		Description
1.	Create a new text asset	Click the **New** and **Text** commands in the pop-up menus to open a blank **Edit Text** window. Use the **Edit Text** window to create a new text asset.
2.	Edit an existing text asset	Click the **Edit** command in the pop-up menu to open an existing text asset in the **Edit Text** window. Use the **Edit Text** window to edit an existing text asset.
3.	**Edit Text** window	Use to create new text assets and edit existing text assets.
4.	Description field	Enter the name of the text asset as you want it to be listed in the **Library** tab.

Screen Editors

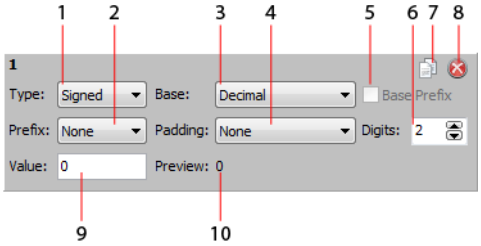
Edit Text window (continued)

Item		Description
5.	**Original Language**	Enter the text and the data values that you want to display. The text that you enter should be in the source language that you use for translation into other languages. For data values, use the **%N$** format, where **N** identifies the number of the Format Data Value panel that you use to format the data value. In the preceding figure, the: — **1** Format Data Value panel **(callout 7a)** formats the **%1$** data value. — **2** Format Data Value panel **(callout 7b)** formats the **%2$** data value.
6.	**Format** tab	**Add** button—opens a Format Data Value panel. **Remove** button—removes the selected Format Data Value panel and the data value that it formats.
7a.	**Format Data Value** tab	Formats the **%1$** data values. The **bold** number in the upper left-hand corner of a Format Data Value panel identifies the data value that it formats.
7b.	**Format Data Value** tab	Formats the **%2$** data values. The **bold** number in the upper left-hand corner of a Format Data Value panel identifies the data value that it formats. See *About Formatting Data Values* on page 463.
8.	Value/Preview field	Enter a value to see the results of the formatting that you apply in the Format Data Value panel.
9.	**Preview** tab	Previews the text as it will appear in a display terminal.
10.	**Font** drop-down list	Sets the font in which the Preview tab displays text. (You set the font that the display actually uses in the **Font** tab of the **Common Properties** window or in the **Inspector** tab.)
11.	**Text Instance**	Selections in this menu become available once you place an instantiation of the text asset in the Design Area. Click the drop-down list to see the text asset as you have instantiated it in the Screen Description. This view shows the font selection that you made in the Font tab of the Common Properties window or in the Inspector tab.
12.	**Translations**	Enter translations of the **Original Language** text here. The **Languages** listed in the **Screen Library** tab's sets the number of Language panels.
13.	**Show Original** check box	Click to alternate the text shown in the **Preview** between the **Original Language** text and a **Translations** text that you select. In the preceding figure, — **Swedish** is the selected Translations text. (Note the darker background.) — click the **Show Original** button to alternate the text shown in the **Preview** tab between the **Original Language (English)** text and **Swedish** text.

Screen Editors

About Formatting Data Values

You format the appearance of data values in the Edit Text window's Data Value panels.

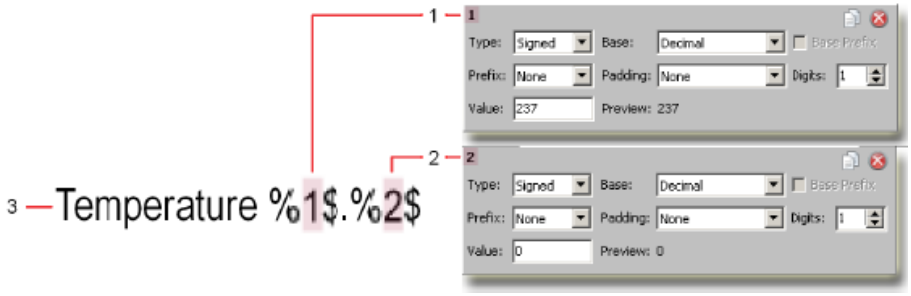


Item	Name	Description		
1.	Type	Select the data type of the data value.		
		Type	**Description**	**Matching data types**
		ASCII	Display an ASCII character in the range 32..127. If the value is outside this range, a space will be displayed.	U8
		BOOL	Display a Boolean value. True is displayed as 1, false is displayed as 0.	BOOL
		Signed	Display a signed integer value.	STRING
		String	Display a string.	STRING
		Unsigned	Display an unsigned integer value.	U8, U16, U32
2.	Prefix	Display a prefix in front of an integer value.		
		Prefix	**Prefix when value >= 0**	**Prefix when value < 0 (signed integers only)**
		None	None	None
		Space	A space	-
		+/-	+	-
		Only relevant if type is set to Signed or Unsigned and Base is Decimal.		
3.	Base	Select which base to use for integer values. Available choices are octal, decimal, hexadecimal with lowercase a..f, hexadecimal with uppercase A..F. When negative signed values are displayed and the base is not decimal, the two's complement will be output. For example: If the data type is S8, the value -1 and the base uppercase hexadecimal, then the displayed value will be FF. Only relevant if type is set to Signed or Unsigned.		

Screen Editors

Item	Name	Description	
4.	Padding	Add padding to make an integer occupy the same minimum amount of space regardless of its value. Minimum space is specified by the Digits property.	
		Padding	**Description**
		None	No padding will be added.
		Left	Spaces will be added to the left of the value.
		Right	Spaces will be added to the right of the value.
		Zero	Zeros will be added to the left of the value.
		Only relevant if type is set to Signed or Unsigned.	
5.	Base Prefix	Display a base prefix.	
		Base	**Prefix**
		Octal	0
		Hex (lowercase)	0x
		Hex (uppercase)	0X
		Only relevant if type is set to Signed or Unsigned and Base is not Decimal.	
6.	Digits	Define how many digits to display when Padding is set to a value different from None. Only relevant if type is set to Signed or Unsigned.	
7.	Copy	Copy the format code placeholder so that it can be pasted into the text boxes.	
8.	Delete	Delete this format code.	
9.	Value	Enter a preview value. Not available for type STRING.	
10.	Preview	Watch how the preview value will be displayed when current settings are applied. Not available for type STRING.	

The following figure shows how these panels format the readout of a Temperature text asset.



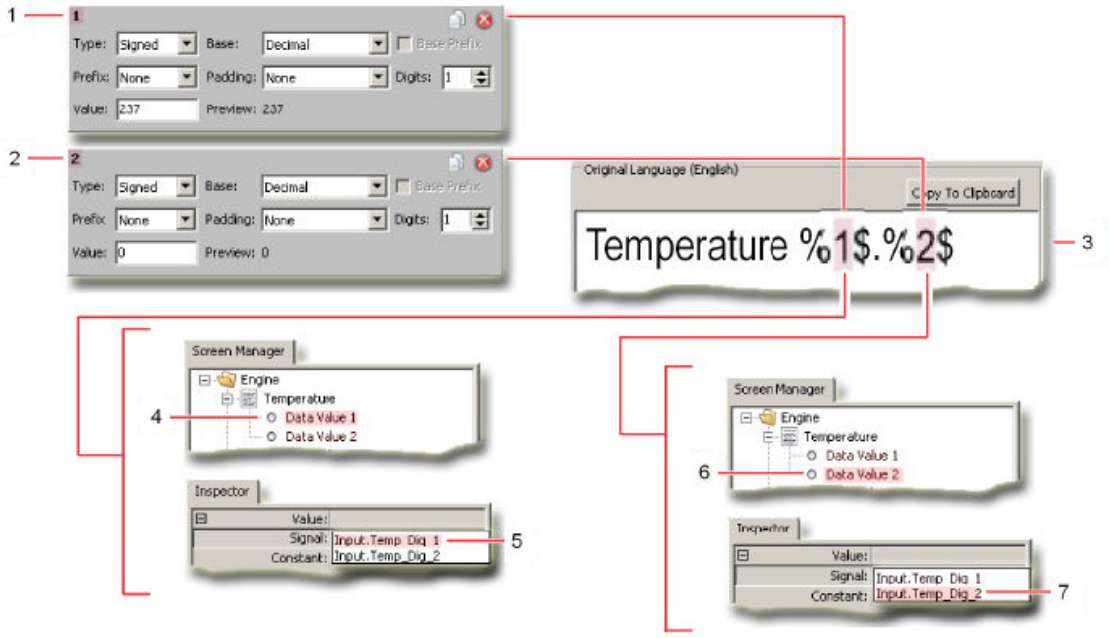
1. Format Data Value panel **1** formats the **%1$** data value.
2. Format Data Value panel **2** formats the **%2$** data value.
3. The text and data values as displayed in the Edit Text window's Original Language panel.

Screen Editors

About Assigning Signals to Data Values

You assign signals to Data Values in the **Screen Editor** window's **Inspector** tab.

The assignment of signals to a Temperature text asset's Data Value 1 and Data Value 2.



1. Format Data Value panel 1 formats the Data Value 1 for the Temperature text asset.
2. Format Data Value panel 2 formats the Data Value 2 for the Temperature text asset.
3. Original Language panel displays the Temperature asset's text and Data Values.
4. Data Value 1 for the Temperature text asset.
5. Input.Temp.Dig_1 signal – the Inspector tab in the Screen Editor window assigns the Input.Temp_Dig_1 signal to Data Value 1.
6. Data Value 2 for the Temperature text asset.
7. Input.Temp.Dig_2 signal – the Inspector tab in the Screen Editor window assigns the Input.Temp_Dig_2 signal to Data Value 2.

About Language Lists and Translations Languages

The following figure shows the relationship between the **Languages** listed in the **Screen Library** tab and the **Translations** languages shown in the **Edit Text** window.

- **English** is the **Original Language**.
- **German**, **Swedish**, and **Russian** are the **Translations** languages.
- A right-click on the **German**, **Swedish**, or **Russian** nodes displays a pop-up menu that lets you set one of these languages as the **Original Language**.

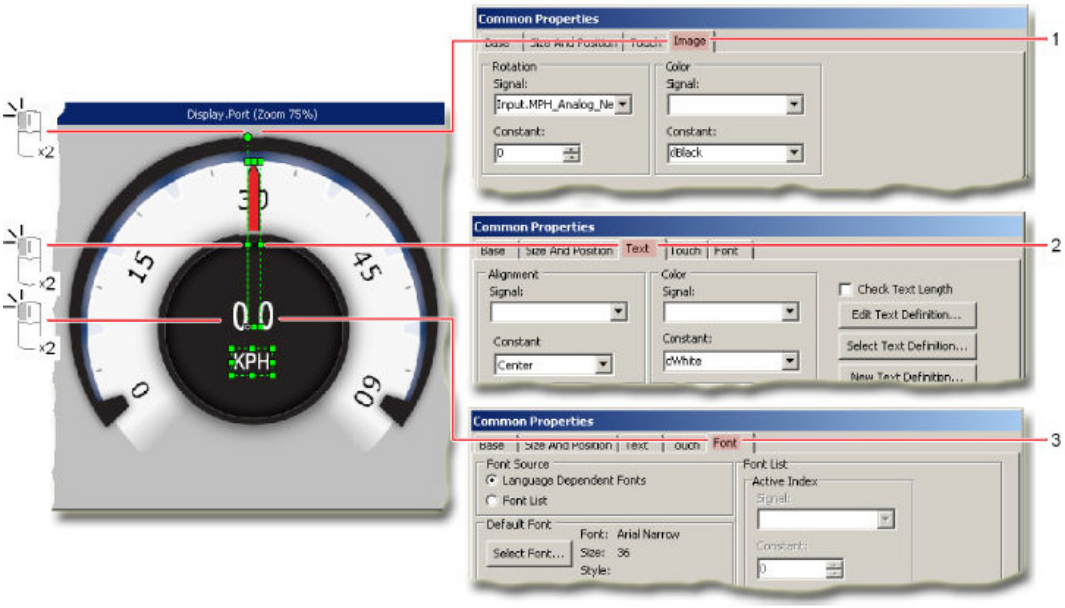
Screen Editors

Common Properties Windows

Use the **Common Properties** windows to assign properties to the screen assets that you instantiate in a **Screen Definition**.

- You can instantiate the same screen asset more than once in the same **Screen Definition** and assign different properties to each instantiation.
- You can instantiate the same screen asset in different **Screen Definitions** and assign different properties to each instantiation.
- The common properties that you can assign when you instantiate a screen asset depend on the type of asset. For example, a text asset has unique **Text** properties and an image asset has unique **Image** properties.

Most of the properties that are available through the **Common Properties** windows are also available through the **Inspector** tab. See *Inspector Tab* on page 474.



1. The **Common Properties** window has an **Image** tab with image-related properties such as **Rotation** and **Color**.
2. The **Common Properties** window has a **Text** tab with text-related properties such as **Alignment** and **Color**.
3. The **Common Properties** window has a **Font** tab with font-related properties such as the **Font Source** and **Default Font**.

Screen Editors

Screen Manager Tab

Use the **Screen Manager** tab to organize and assign signals to the screen assets that you place in the Design Area.

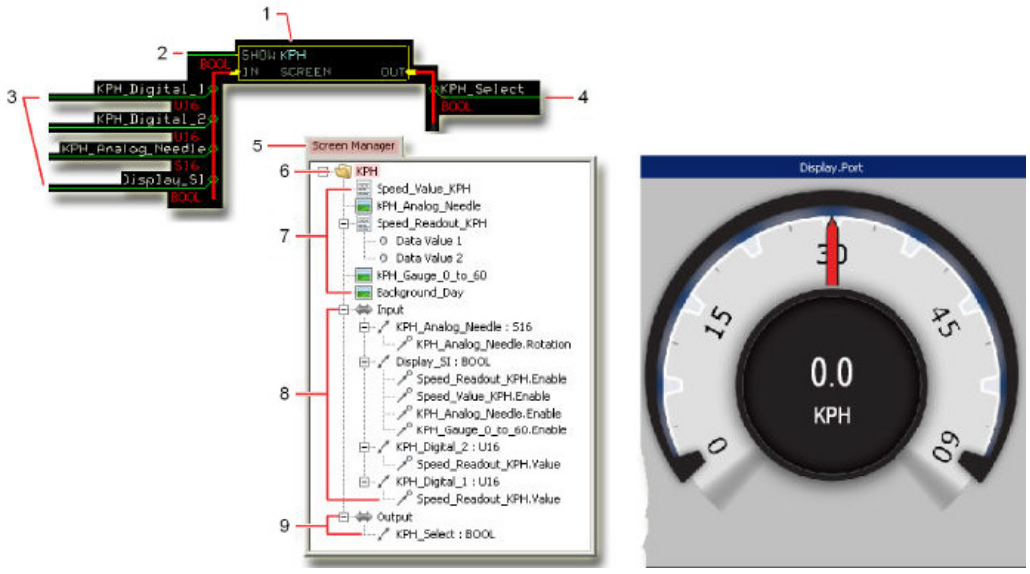
The **Screen Manager** tab lists screen assets in front-to-back order. You can click-and-drag in this tab to rearrange the order of screen assets. You make signals available in the **Screen Manager** tab for assignment to screen assets in two ways:

1. Manually, by clicking and typing in the **Screen Manager** tab. See *How to Manually Make Signals Available for Assignment to Screen Assets* on page 471.
2. Automatically, through the **Query Screen Component** window. This is the faster and more accurate method. See *About the Show Screen Component and the Query Screen Component Window* on page 510.

Signal connections between a screen definition, widgets and POU calls are described in topic *Internal Connections* on page 496.

KPH Screen Definition

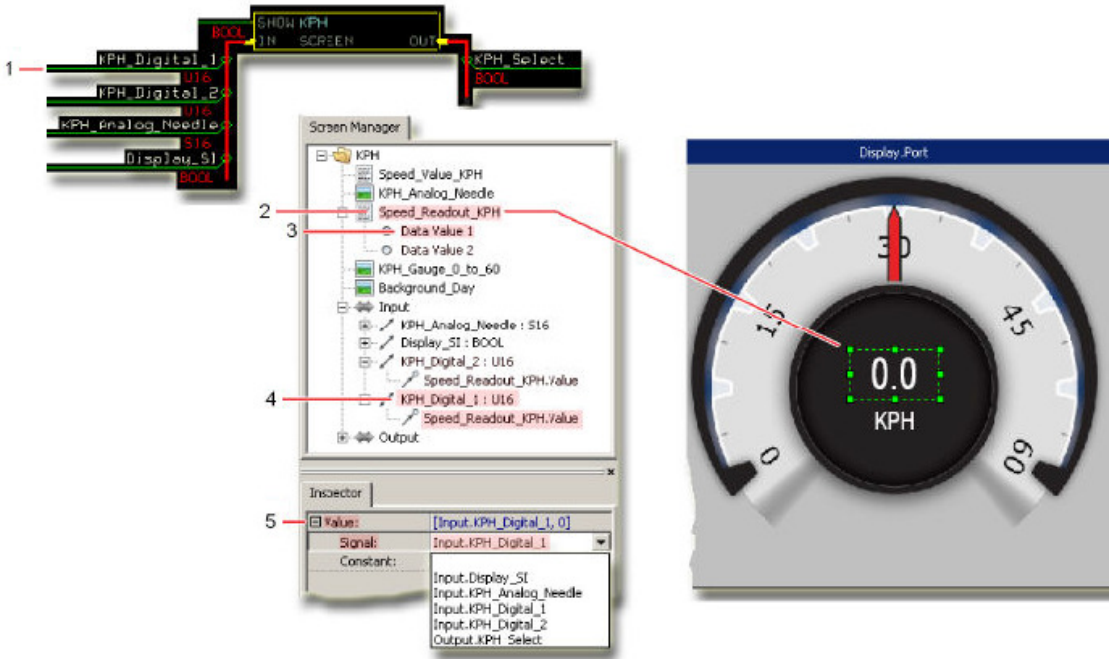
The following figure shows the relationship between the signals that connect to a **Show Screen** component and the contents of the **Screen Manager** tab in the **Screen Definition** that the Show Screen component instances.



1. **Show Screen** component instances the **KPH Screen Definition** .
2. **Show** input enables the display of the entire **KPH Screen Definition** .
3. Input signals control the display of assets in the **KPH Screen Definition** .
4. Output signal goes true when a selected asset in the **KPH Screen Definition** becomes visible.
5. **Screen Manager** tab for the **KPH Screen Definition** .
6. Use the **Inspector** tab to change Screen Definition name.
7. **Screen Assets** lists the screen assets used in the Design Area for the**KPH Screen Definition** . Assets list top-down in the front-to-back order in which they display in the Design Area.
8. Input signals list the:
   - signals available to the Show Screen component that instances the **KPH Screen Definition** .
   - assignment of signals to assets in the Design Area. Listed signals can remain unassigned.
9. Output signal lists signals output from the Show Screen component that instances the **KPH Screen Definition** .

Screen Editors

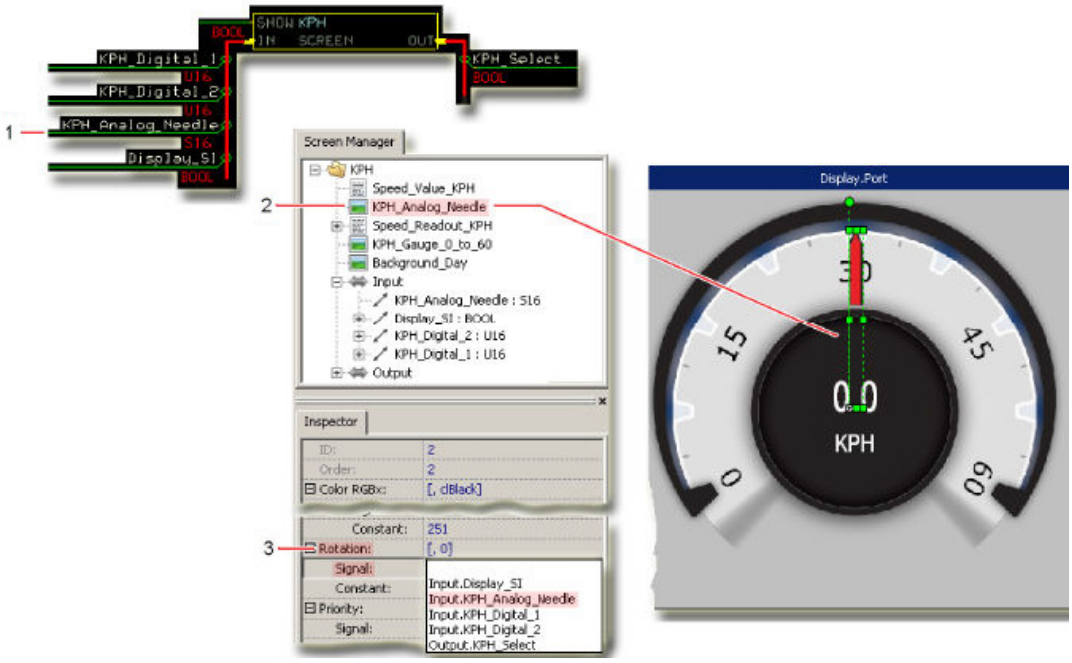
The signal configuration for a text asset with a Data Value



1. **KPH_Digital_1** signal Inputs the integer (whole) kilometers-per-hour value. (The **KPH_Digital_2** signal inputs the fractional kilometers-per-hour value.)
2. **Speed_Readout_KPH** text asset displays integer (whole) and fractional kilometers-per-hour through Data Value 1 and Data Value 2. (You define and format data values when you create a text asset in the Edit Text window.)
3. The signal assigned to **Data Value 1** inputs the integer kilometers-per-hour value. (The signal assigned to **Data Value 2** inputs the fractional kilometers-per-hour value.)
   - Click a Data Value to assign a signal to input a value
   - Use the Inspector tab's Value property to assign the signal.
4. Data Value 1 signal assignment shows the KPH_Digital_1 signal's assignment to the Speed_Read_KPH text asset's Data Value 1.
5. **Value** property signal assignment assigns the signal that inputs the integer kilometers-per-hour value to the Speed_Read_KPH text asset's Data Value 1.

Screen Editors

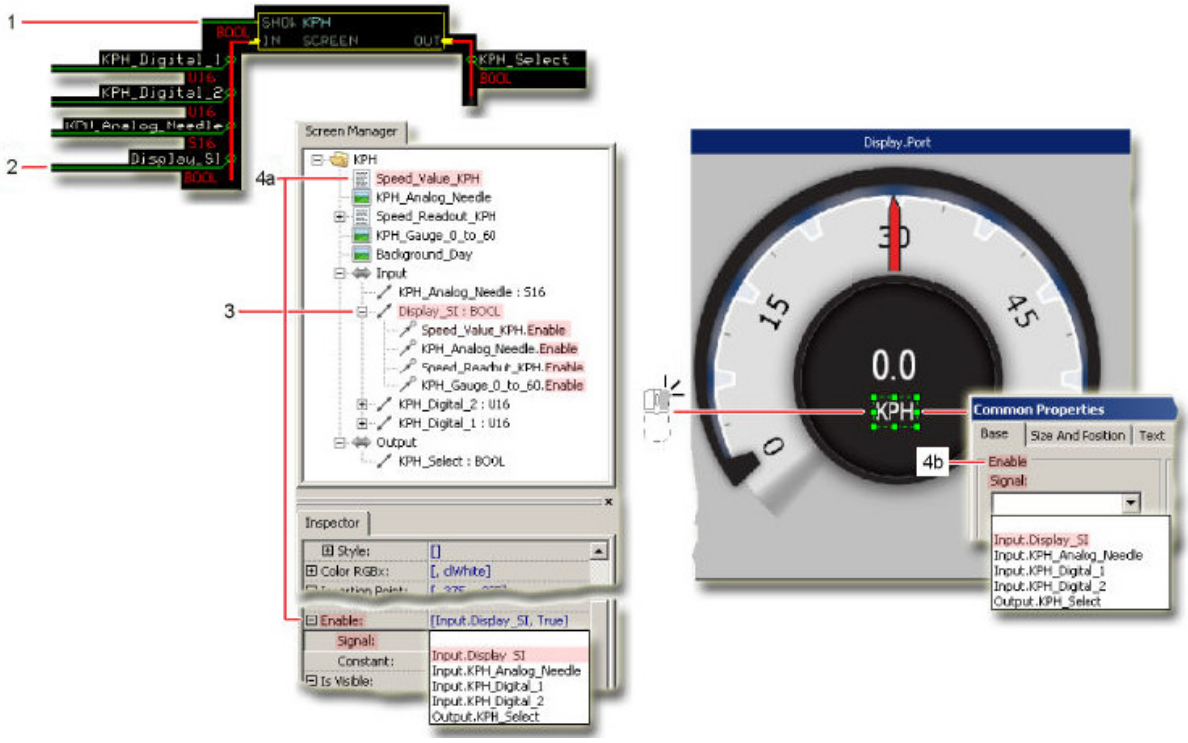
The signal configuration for an image asset with a Rotation property



1. **KPH_Analog_Needle** signal inputs a rotational value to indicate kilometers-per-hour.
2. **KPH_Analog_Needle** image asset rotates to indicate kilometers-per-hour.
3. **Rotation** property signal assignment selects the signal that rotates the KPH_Analog_Needle image asset.
   - Click an image asset to assign a signal to rotate the image asset.
   - Use the **Inspector** tab's Rotation property to assign the signal that rotates the asset.

Screen Editors

The signal configuration for screen assets with an Enable property



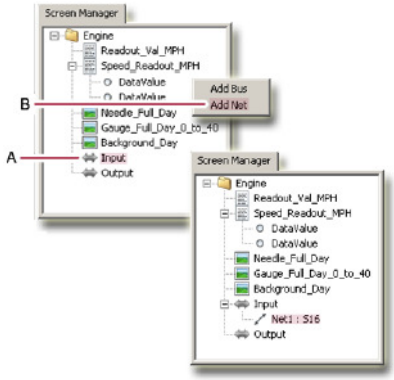
1. The Boolean **Show** input signal must be true to enable the display of the entire **KPH Screen Definition**.
2. The Boolean **Display_SI** signal when true displays the:
   - **Speed_Value_KPH** text asset
   - **KPH_Analog_Needle** image asset
   - **Speed_Readout_KPH** text asset
   - **KPH_Gauge_0_to_60** image asset
3. The Boolean **Display_SI** signal assignment show the **Enable** property of the:
   - **Speed_Value_KPH** text asset
   - **KPH_Analog_Needle** image asset
   - **Speed_Readout_KPH** text asset
   - **KPH_Gauge_0_to_60** image asset
4. **a, b Enable** property signal assignment assigns the signal that when true enables the display of a screen asset. You can assign a signal to an asset's Enable property through either the **Inspector** tab or through the **Common Properties** window's **Base** tab.

Screen Editors

How to Manually Make Signals Available for Assignment to Screen Assets

The following steps describes the manual click-and-type method. (The steps to add a bus are similar.)

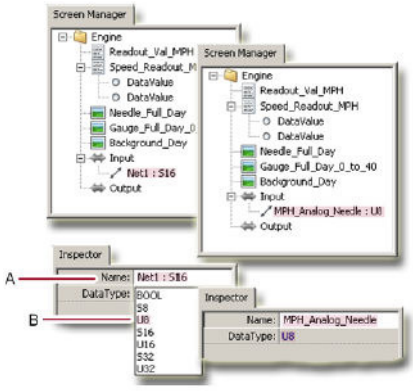
1. Make a signal available for assignment.



A Click the desired type of node (**Input** or **Output**) in the **Screen Manager** tab, to open a pop-up menu.

B Click **Add Net** in the pop-up menu that opens, to add a signal.

2. Name the new signal and set its data type.



A Name the added signal in the **Inspector** tab.

B Select the new signal's **DataType**.

For a "through connection" the names and data types of signals added to the **Screen Manager** tab must match the names and data types of signals connected to the instancing **Show Screen** component's **IN** and **OUT** buses.

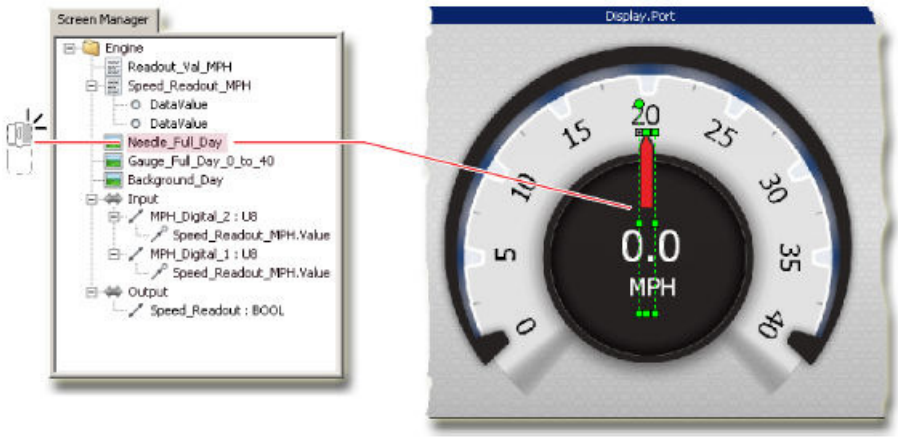
Screen Editors

How to Assign an Available Signal to a Screen Asset

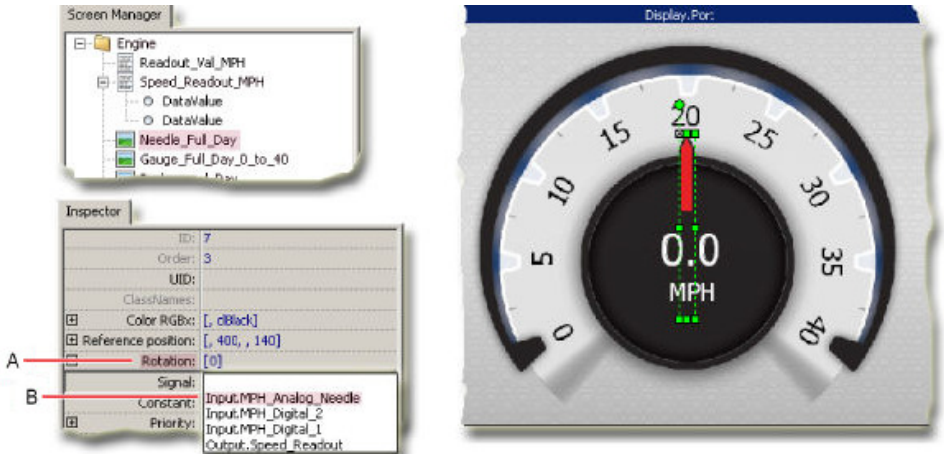
Most signals can also be assigned to a screen asset through a screen asset's **Common Properties** windows. See *Common Properties Windows* on page 466.

1. Click the screen asset to which you want to assign a signal.

   The **Design Area** highlights the selected screen asset.

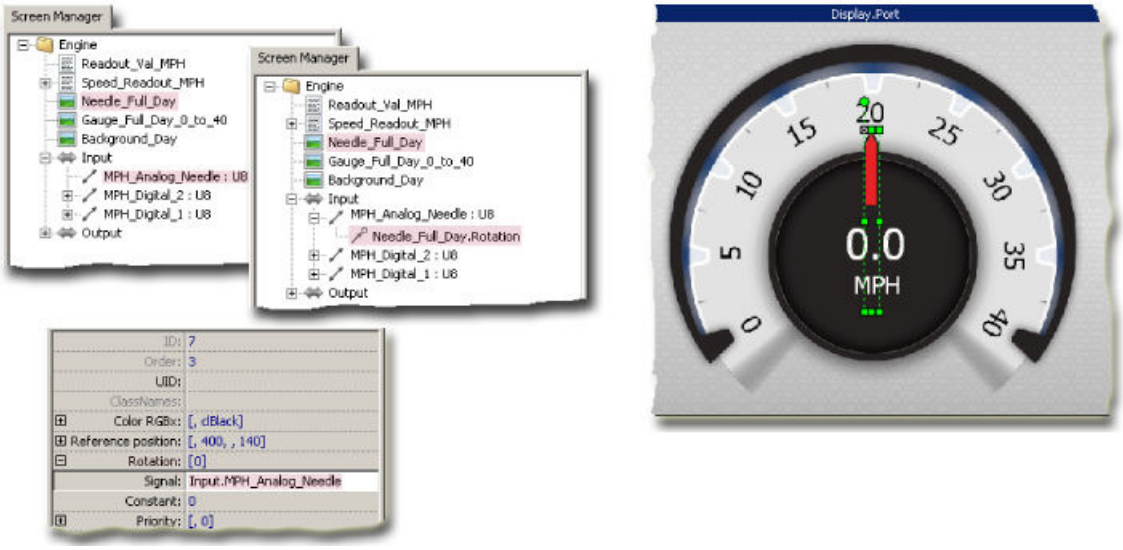


2. Click the property of the screen asset in the **Inspector** tab and select the signal, that you want to manage.

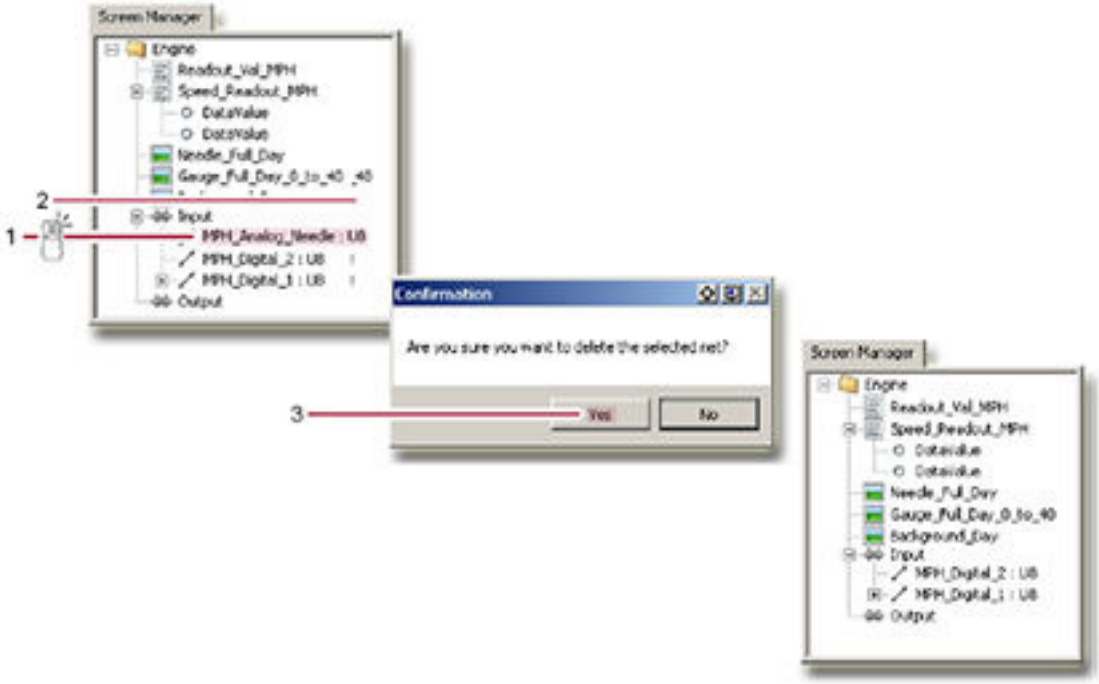


The **Screen Manager** tab shows the signal assignment.

Screen Editors



How to Delete a Signal from a Screen Definition



1. Right-click the signal node in the **Screen Manager** tab, that you want to delete.
2. Click **Delete** in the pop-up menu.
3. Click **Yes** in the **Confirmation** window.

The signal from a Screen Definition in the **Screen Manager** tab was deleted.

Screen Editors

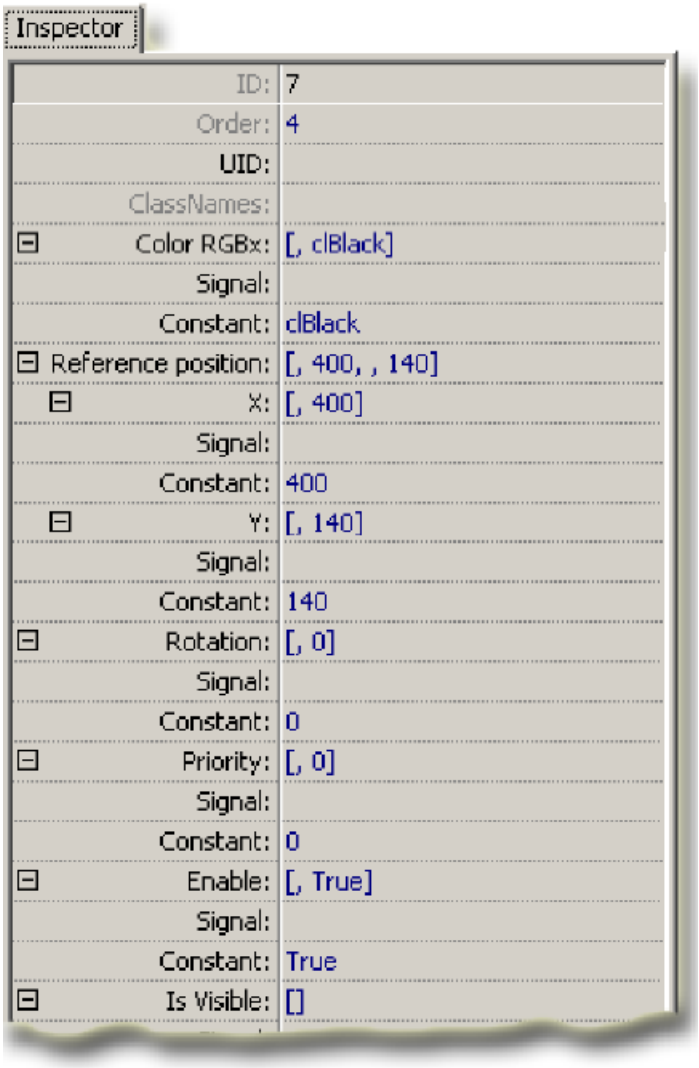
Inspector Tab

Use the **Inspector** tab to review and manage the properties of screen assets placed in the Design Area.

Most of the properties that are available through the **Inspector** tab are also available through the **Common Properties** window. See *Common Properties Windows* on page 466 .

You can view the properties of a screen asset two ways:

- In the **Design Area**, click the screen asset itself.
- In the **Inspector** tab, click the name of the screen asset.



Different types of screen assets have different properties. The screen asset that you select determines the properties that the **Inspector** tab displays and the properties that you can change.

The tables below describe the properties of the screen objects.

Screen Editors

Common properties

Property	Description	Screen Object Types
Name	This property specifies the name of the screen objects. The Name property does not have any significance in the compiled application, its purpose is to enable the developer to give proper names to screen objects. It does not have to be unique, although unique names are strongly recommended.	All
Description	The description of the referenced screen library definition, if any, or the type of the screen object.	All
Layout	Layout properties. See *Layout* on page 491.	All
Order	Sets the front-to-back display order of screen assets. For example, when you simultaneously enable the display of two screen assets, the screen asset with an **Order** property of **0** displays in front of the screen asset with an **Order** property of **1**. The sequence in which you drag a screen asset into the Design Area sets its initial **Order** property. You can change the **Order** property of a screen asset in two ways: • Click and drag in the **Screen Manager** tab to change the order in which this tab lists screen assets. • **Move Forward** • **Move Backward**, **Bring to Front**, and **Send to Back**. • **Right-click a screen asset in the Design Area** to open a pop-up menu with commands.	Text, Text List, Image, Image List, Line, Hardware Image, Widget, Generic Viewport
ID	Right-click a screen asset in the Design Area to open a pop-up menuID of referenced screen library object or hardware image. This is a read-only property.	Text, Text List, Image, Image List, Hardware Image, Widget
GUID	Unique ID of a screen object. This is a read-only property.	All
Enable	Enables the display and touch detection of a screen object. (**Touch Areas** will always be invisible when an application is run.)	All
Color RGBx	Defines the color of a screen object or screen definition. Possible values include a set of predefined colors: **Transparent** which will set the color to fully transparent, and **Custom Color…** which will open a dialog where the color can be freely defined. Double-clicking the **Constant combobox** will open the custom color dialog with the current color preselected. This may be useful for finding out the RGB code of a certain predefined color. Setting the color of an **Image** object or an image at the currently active index of an **Image List** will not have any effect unless the image is of **Monochrome (1 bit)** image format.	Text, Text List, Image, Image List, Line
Priority	Determines which screen asset displays when two or more assets: • Have different Priority values. (All screen assets start with a default Priority value of 0.) • Have an Enable value of True. Only the screen asset with the highest Priority value displays. The lower the Priority number, the higher the display priority. A Priority value of 0 is the highest display priority.	Text, Text List, Image, Image List, Line, Hardware Image, Generic Viewport
IsVisible	Assign a BOOL output signal to this property. If this signal: • True — the screen asset is visible. • False — the screen asset is not visible.	Text, Text List, Image, Image List, Line, Hardware Image, Generic Viewport
ActiveIndex	The list element at this index will be displayed. If ActiveIndex is outside the range of valid indexes, the first or last element will be displayed.	Image List, Text List
FlipHorizontal	Flip an image in the horizontal dimension.	Image, Image List
FlipVertical	Flip an image in the vertical dimension.	Image, Image List, Hardware Image
FontEx	Font attributes of a text. **Use Font List** must be false in order to use **FontEx**.	Text, Text List

Screen Editors

Common properties (continued)

Property	Description	Screen Object Types
CheckTextLength	Generate a compile error if a text object is too small to contain its referenced screen library text.	Text, Text List
Alignment	Text justification. Possible values are Left, Center, Right.	Text, Text List
LDFList	Manage the list of language dependent fonts. This list contains pairs of languages and fonts. If the currently active language has a matching font in the list, that font is used. Otherwise, the font specified by **FontEx** will be used. **Use Font List** must be false in order to use **LDFList**.	Text, Text List
Use Font List	Use the font specified by **Font List** instead of **FontEx**.	Text, Text List
Font List	Add fonts to a list and select which font to use with the Active Index Signal. In order to use **Font List**, **Use Font List** must be set to true.	Text, Text List
Touch	Touch properties. See *Touch Display Functionality* on page 524	

Line properties

Property	Description
Width	Sets the width, in pixels, of a **Line** object.

Image properties

Property	Description
Image Definition	Screen library asset referenced by an image object. The constant value may be undefined (indicated by the value -1) if the signal is connected to the screen definition interface and the screen definition is being used as a widget (see *Screen Definitions and Widgets* on page 487). If the constant value is undefined and the parent screen definition is referenced by a Show Screen component, the PLUS+1® GUIDE project will not compile.

Text properties

Property	Description
Text Definition	Screen library asset referenced by a text object. The constant value may be undefined (indicated by the value -1) if the signal is connected to the screen definition interface and the screen definition is being used as a widget (see *Screen Definitions and Widgets* on page 487). If the constant value is undefined and the parent screen definition is referenced by a Show Screen component, the PLUS+1® GUIDE project will not compile.

Image list properties

Property	Description
Image size	Defines how the images in the list will be resized. **Aspect Ratio Preserved** – each image will as large as possible within the boundaries of the image list, with preserved aspect ratio. **Scale Image to Image List Size** – Each image in the list will be as large as the image list, ignoring its aspect ratio.
Image List Definition	Screen library asset referenced by an image list object. The constant value may be undefined (indicated by the value -1) if the signal is connected to the screen definition interface and the screen definition is being used as a widget (see *Screen Definitions and Widgets* on page 487). If the constant value is undefined and the parent screen definition is referenced by a **Show Screen** component, the PLUS+1® GUIDE project will not compile.

Screen Editors

Test list properties

Property	Description
Text List Definition	Screen library asset referenced by a text list object. The constant value may be undefined (indicated by the value -1) if the signal is connected to the screen definition interface and the screen definition is being used as a widget (see *Screen Definitions and Widgets* on page 487). If the constant value is undefined and the parent screen definition is referenced by a **Show Screen** component, the PLUS+1® GUIDE project will not compile.

Hardware image properties

Property	Description
AlwaysOnTop	The hardware image will be rendered in overlay mode if possible. Refer to the API specification of the HWD for more information.

Generic viewport properties

Property	Description
Outputs	The properties in this branch output the values set in the **Layout** branch. Use them when these properties need to be known at run-time.
UIDOutput	A unique number identifying the generic viewport object. Connect it to the corresponding graphics source in the SYS.

Toolbar

Toolbar buttons description

Button		Description
	Save	Save all screen editor related data.
	Undo	Undo operations.
	Redo	Redo operations.
	Cut	Cut selected item(s).
	Copy	Copy selected item(s).
	Paste	Paste cut/copied item(s).
	Manager	Toggle Screen Manager and Definitions panel.
	Inspector	Toggle Inspector panel.
	Screen Library	Toggle Screen Library panel.
	Selector	Toggle Selector panel.
	Zoom In	Zoom in the design area.
	Zoom Out	Zoom out the design area.
	Search	Open the Search/Replace dialog.
	Preview	Display the design area in preview mode.

Screen Editors

Toolbar buttons description (continued)

Button		Description
	Show/Hide Skin	Toggle hardware skin. If shown, the hardware skin will show how the front of the display unit will align with the graphics on the display.
Standard Buttons	**Standard Buttons**	Select which skin to show when Show/Hide Skin is active. It will only be possible to select skin when the HWD has more than one skin to choose from.
	Pan With Hand	Select pan mode. See chapter *Pan and Zoom* on page 481 for more details.
	View and Delete Unused Library Items	Open a dialog where unused screen library items can be deleted. (Note that unused screen library items will not be included in the compiled application.)
{a}	**Edit Code Point Set**	Open the Code Point Set dialog where glyphs to include for string rendering are defined. (Note that this dialog is only needed when string signals are connected to text objects.)
Aa	**Edit Font Output Formats**	Select which font output formats to use. This functionality is hardware dependent.
	Exit Screen Editor	Close Vector-Based Screen Editor.

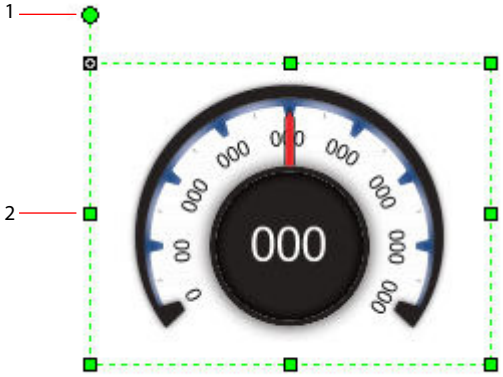
Design Area

Use the design area to edit and move screen objects. Alternatively, screen objects can be selected in the Screen Manager and edited in the Inspector. Screen objects can be moved using the mouse or arrow keys. While using the arrow keys, holding down **Ctrl** will give enhanced precision.

The screen definition itself can be resized and repositioned by dragging its edges. Holding down **Ctrl** while dragging the screen definition will reposition it without changing its size.

Holding down the Ctrl key while performing a mouse operation in the Design Area will sometimes modify the operation.

Clicking a screen object will select it. If allowed, a selected screen object can be rotated or resized using the mouse.



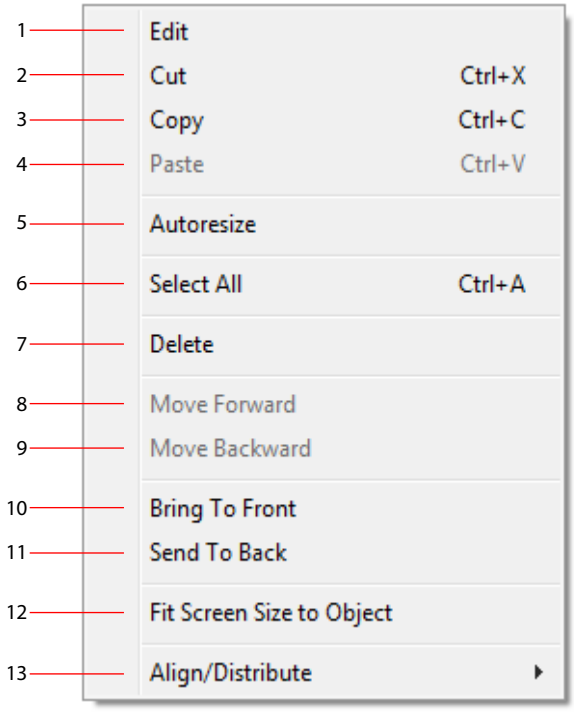
Screen Editors

Item	Name	Description
1	Rotation handle	Press the left mouse button above this handle and move the mouse while the button is pressed to rotate the screen object. Holding down the **Ctrl** key will rotate with increments of 15°. Screen objects that cannot be rotated do not have a rotation handle.
2	Resize handle	Press the left mouse button above this handle and move the mouse while the button is pressed to resize the screen object. If the screen object is setup to have preserved aspect ratio, its width and height cannot be changed independently. Screen objects that cannot be resized have gray resize handles. Line objects have two resize handles, one for each end point. Holding down the **Ctrl** key while moving a line end point will force the line to be either horizontal or vertical.

Multiple screen objects can be selected by holding down the **Shift** key while clicking them or by pressing the left mouse button above the design area and keeping it down while drawing an area. When the mouse button is released, all screen objects within the drawn area will be selected.

When multiple screen objects are selected, properties they have in common can be changed at the same time in the Inspector. They can also be aligned and distributed (see *Alignment and Distribution* on page 480).

Design Area Context Menu

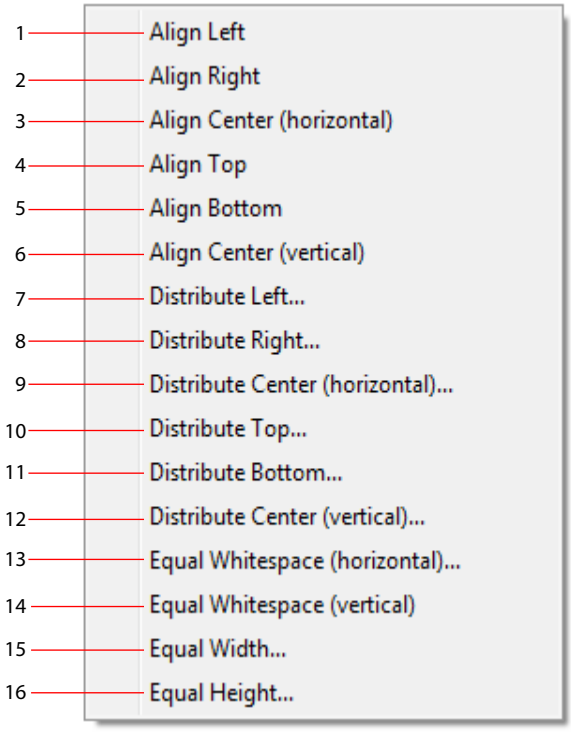


Item	Name	Description
1	Edit	Open the Common Properties window of a screen object.
2	Cut	Cut selected screen objects.
3	Copy	Copy selected screen objects.
4	Paste	Paste copied screen objects.
5	Autoresize	Resize a screen object to its default size.
6	Select All	Select all screen objects in the current screen definition.
7	Delete	Delete selected screen objects.
8	Move Forward	Move a screen object forward in the front-to-back display order.

Screen Editors

Item	Name	Description
9	Move Backward	Move a screen object backward in the front-to-back display order.
10	Bring to Front	Move a screen object to the front in the front-to-back display order.
11	Send to Back	Send a screen object to the back in the front-to-back display order.
12	Fix Screen Size to Object	Resize the screen definition to the size of the screen object. The screen object will be moved unless it is already in a position where there is no excess space to the left or above it.
13	Align/Distribute	Open the Align/Distribute sub-menu.

Alignment and Distribution



Item	Name	Description
1	Align Left	The left edge of all selected objects will get the same X coordinate as the left edge of the leftmost object.
2	Align Right	The right edge of all selected objects will get the same X coordinate as the right edge of the rightmost object.
3	Align Center (horizontal)	The horizontal center of all selected objects will get the same X coordinate as the center of the smallest rectangle encompassing all selected objects.
4	Align Top	The top edge of all selected objects will get the same Y coordinate as the top edge of the top object.
5	Align Bottom	The bottom edge of all selected objects will get the same Y coordinate as the bottom edge of the bottom object.
6	Align Center (vertical)	The vertical center of all selected objects will get the same Y coordinate as the center of the smallest rectangle encompassing all selected objects.
7	Distribute Left…	All selected objects will be distributed with a specified number of pixels between their left edges. The leftmost object will not move and the left-to-right order will be preserved.
8	Distribute Right…	All selected objects will be distributed with a specified number of pixels between their right edges. The rightmost object will not move and the left-to-right order will be preserved.

Screen Editors

Item	Name	Description
9	Distribute Center (horizontal)	All selected objects will be distributed with a specified number of pixels between their horizontal centers. The objects will be centered horizontally in the smallest rectangle encompassing all selected objects and the left-to-right order will be preserved.
10	Distribute Top…	All selected objects will be distributed with a specified number of pixels between their top edges. The top object will not move and the top-to-bottom order will be preserved.
11	Distribute Bottom…	All selected objects will be distributed with a specified number of pixels between their bottom edges. The bottom object will not move and the top-to-bottom order will be preserved.
12	Distribute Center (vertical)…	All selected objects will be distributed with a specified number of pixels between their vertical centers. The objects will be centered vertically in the smallest rectangle encompassing all selected objects and the top-to-bottom order will be preserved.
13	Equal Whitespace (horizontal)…	All selected objects will be distributed horizontally with a specified number of pixels them. The leftmost object will not move and the left-to-right order will be preserved.
14	Equal Whitespace (vertical)…	All selected objects will be distributed vertically with a specified number of pixels them. The top object will not move and the top-to-bottom order will be preserved.
15	Equal Width…	All selected objects will get a specified width.
16	Equal Height…	All selected objects will get a specified height.

Pan and Zoom

If the toolbar button **Pan with Hand** is active:

- Press and hold the middle or right mouse button to pan the design area by dragging the mouse.
- No scroll-bars will be available.

Use toolbar buttons or the mouse wheel to zoom. The zoom behavior depends on the **Pan with Hand** toolbar button:

- If **Pan with Hand** is **active**, the zoom will be centered at the mouse pointer.
- If **Pan with Hand** is **not active**, the top left area of the design area will be zoomed.

The **Zoom Fit Page** command (default key **HOME**) will set zoom level to 100% and center the **Design Area**.

Data Types

Data types are specified for the screen definition interface signals. The type of these signals must match the data types of the corresponding signals connected to the buses of the Show Screen component. When a screen definition is being used as a widget; its interface signals, if any, are shown in the Screen Manager. (See *Screen Definitions and Widgets* on page 487).

POU Calls inside a screen definition shows the PLUS+1® GUIDE data types corresponding to the IEC61131 data types added in the PLC Editor. (See *About PLC Data Types* on page 387 and *Call POU from screen* on page 495).

Some screen object properties can be connected to signals. The data types of screen editor objects are implicit and not shown in the inspector.

Finally, text object data values also have data types that are defined in the corresponding text definition. These data types are not shown in the inspector.

Integer, Boolean and Color

The following data types in these categories are supported in Vector-Based Screen Editor:

- BOOL
- U8
- S8
- U16
- S16
- U32
- S32
- COLOR

See *Data Types* on page 180 for a detailed list of PLUS+1® GUIDE data types.

Floating-point and 64-bit data types

Data types F32, F64, U64 and S64 may be used in the screen definition interface and as connections to POUs. There is no support for these data types in general in Vector-Based Screen Editor, and it is not possible to enter constant values for them. Hence, any POU or widget input of these data types must originate from another POU or the screen definition interface.

Array Data Types

Arrays may be used in the screen definition interface and as connections to POUs. The following array types are supported:

- BOOL
- U8
- S8
- U16
- S16
- U32
- S32
- U64
- S64
- COLOR
- F32
- F64

There is no support for array data types in general in Vector-Based Screen Editor. Constant values in widget and POU inputs may be entered for arrays of the following types:

- BOOL
- U8
- S8
- U16
- S16
- U32
- S32

A constant array value is entered as a comma-separated list of values.

Internal Data Types

Data types Text, Image, Text List and Image List represent definitions in Screen Library. Signals of these data types may be connected to a screen object, a screen definition interface, and a widget instance. This

allows separation of the actual text or image content from the widget design and may be used to increase reusability.

These data types are internal to Vector-Based Screen Editor and not available elsewhere in PLUS+1® GUIDE. A screen definition having signals of these data types in its interface may only be used as a widget instance. Selecting such a screen definition in a Show Screen component is not allowed.

String

String signals in Vector-Based Screen Editor may originate from a screen definition interface signal or a POU call. They can be rendered on the display using text definitions with data values of type String. Regarding text data values see *About Formatting Data Values* on page 463 and information on how to connect signals to text objects see *About Assigning Signals to Data Values* on page 465.

The maximum number of bytes that can be rendered on the display in one string is specified by the default string length attribute. Regarding information about default string length and the STRING data type in general, see *STRING Types* on page 174.

Text and String Rendering

When a PLUS+1® GUIDE project is being compiled, all glyphs needed to render all text definitions in use will be generated. The ASCII code point set (0-127) will always be included, to ensure rendering of non-string text data values (such as. integers, minus sign, ASCII characters). Constant text data values of type string will also be included automatically.

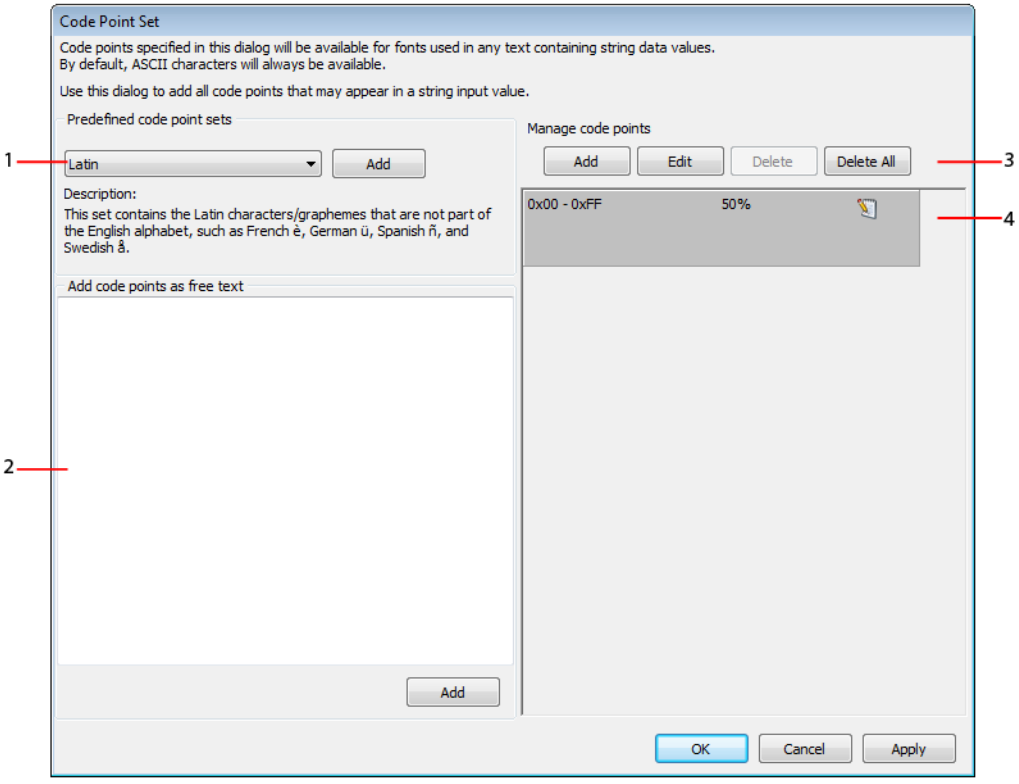
String signals connected to text data values need to be handled manually. This is done in the **Code Point Set** of the PLUS+1® GUIDE project. All code points in the Code Point Set will be available for all text definitions in use that have at least one string data value. Note that code points inside a string are always rendered one by one. For some alphabets, this will result in string rendering that differs from normally expected output.

The Code Point Set consists of ranges of Unicode code points. Additional code points can be added as a predefined set, by typing the desired code points in a text field, and by specifying ranges of code points. Open the Code Point Set dialog by pressing in the toolbar the **Edit Code Point Set** button in the toolbar.

Edit Code Point Set button

Screen Editors

Code Point Set

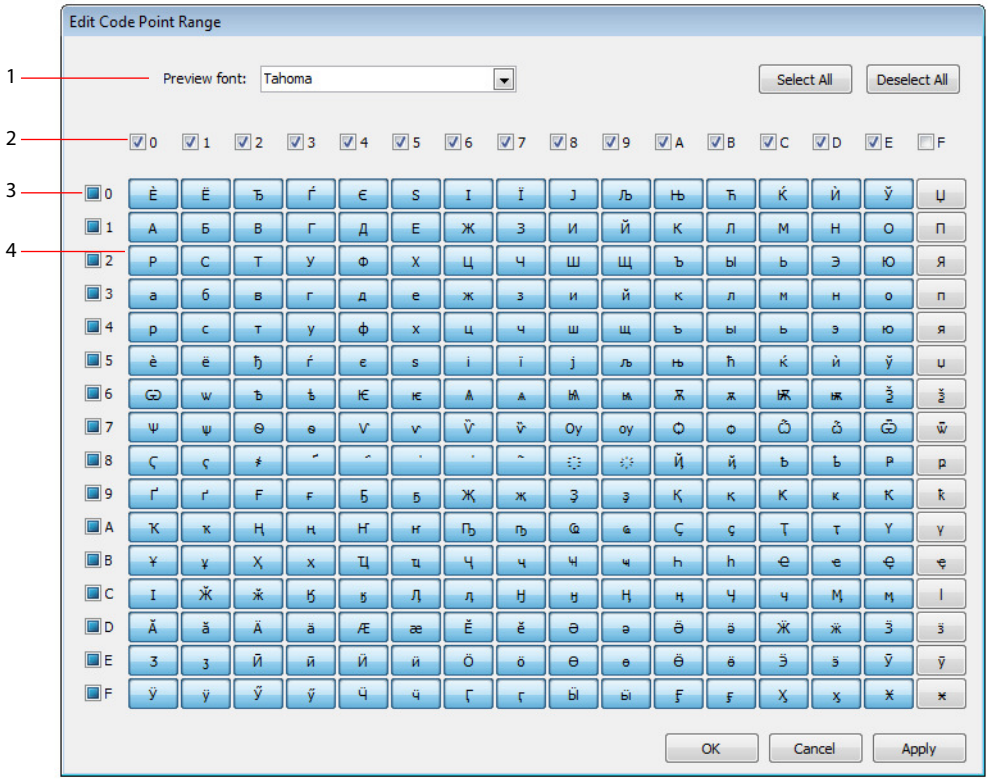


Code Point Set dialog

Item	Name	Description
1.	Predefined code point sets	Choose a predefined code point set that contains all code points needed for an entire alphabet or a set of alphabets to **Add**.
2.	Free text input	Type or paste the text of all code points to **Add** text to the code point set.
3.	Manage code points	**Add** or manage a range of code points.
4.	Code point ranges	A list view of code point sub-ranges. Each sub-range contains 256 code points. Each item in the list shows start and end of the range, and the percent of code points in the range that are included in the code point set. There is also an **Edit** button to open the **Edit Code Point Range** dialog and a **Delete** button to remove the range. Note, it is impossible to delete the first range, because it contains the ASCII code points which will always be included.

Screen Editors

Edit Code Point Range



Edit Code Point Range dialog

Item	Name	Description
1.	Preview font	Control the font to use for the toggle buttons. Note that some fonts may not contain glyphs for all code points.
2.	Column check boxes	Toggle selection of a column.
3.	Row check boxes	Toggle selection of a row.
4.	Code point toggle buttons	Select or deselect individual code points.

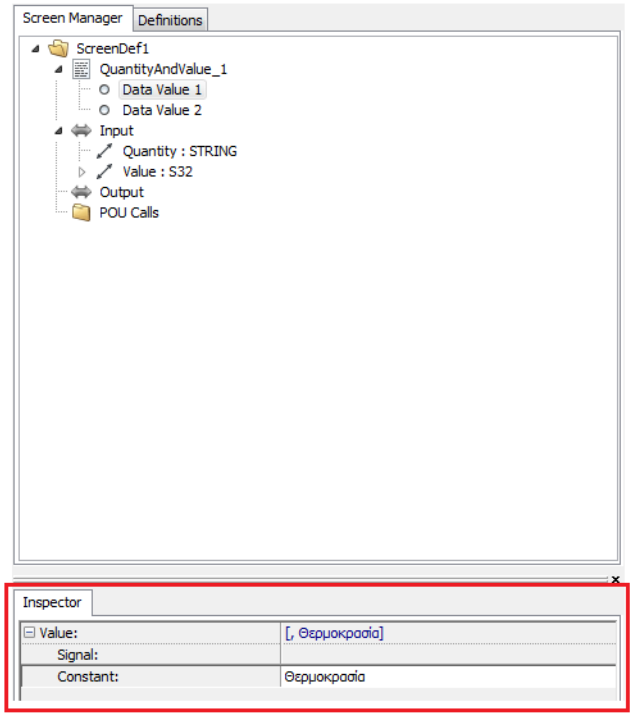
Example

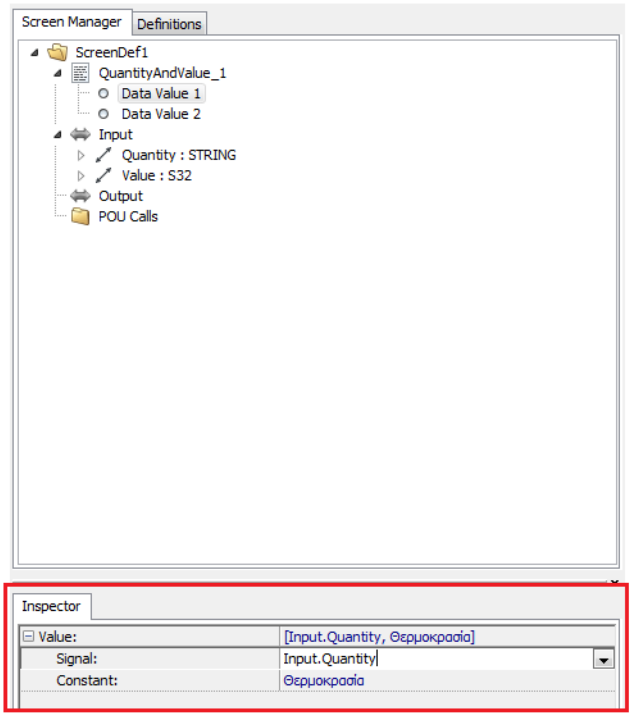
The left side of the following image, a string constant “Θερμοκρασία” is entered as the first data value of text object QuantityAndValue_1. When the application is compiled, Greek characters {Θ, α, ε, ί, κ, μ, ο, ρ, σ} will be included for the font used by QuantityAndValue_1.

The right side of the image, a screen definition input signal Quantity is connected to the first data value of text object QuantityAndValue_1. As a result, string constant “Θερμοκρασία” will not be displayed and no Greek characters will be included automatically for the font used by QuantityAndValue_1.

To ensure proper display of all possible Greek texts, include the Greek range of Unicode characters in the **Code Point Set** dialog.

Screen Editors





Control Codes

Some code points are control codes. These code points will be handled as specified in the following table.

Control code	Effect
0	Rendering of the string will be terminated.
10	Rendering of the string will start on a new line, if it fits inside the text box.
All other control codes < 32	No effect.
All control codes >= 32	The corresponding glyph will be output if it exists and has a width greater than zero. For details, see *Zero-width Glyphs* on page 486 and *Missing Glyphs* on page 486.

Zero-width Glyphs

Glyphs of zero width will not be explicitly output on the display.

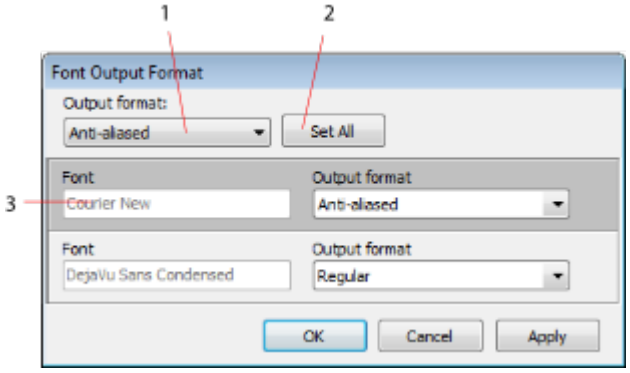
Missing Glyphs

Missing glyphs will be rendered as whitespace (code point 32). A glyph may be missing if its corresponding code point originates from a string connected to a text data value, and that code point was not added to the code point set of the application.

Font Output Format

Use the **Edit Font Output Formats** dialog to control how the glyphs of a font are rendered on the target display. This can be used to turn anti-aliasing on or off. Open the dialog by pressing the **Edit Font Output Formats** button in the toolbar. Aa

Screen Editors



Item	Name	Description
1.	Selection	Select a font output format to apply to all fonts.
2.	Set All	Apply the selected font output format to all fonts.
3.	Individual selection	Apply font output formats individually.

Screen Definitions and Widgets

Screen Definitions may be called from an SCS module using a **Show Screen** component or embedded into other screen definitions. In the latter case, the screen definition that is embedded into another screen definition is referred to as a **widget**.

If screen definition A is added to screen definition B as widget A_1, then A_1 is a **widget instance** of screen definition A. A_1 is referencing screen definition A and screen definition A is the **widget definition** of A_1. Screen definition called from a Show Screen component in an SCS module is referred to as a **top level screen definition**.

Screen definition A can be added to screen definition B if B is not A and A does not contain an instance of B.

It is not possible to edit the internal components of a widget directly; the widget definition must be modified instead. Editing a screen definition will affect all its widget instances. Widgets may be nested.

Use widgets to encapsulate screen objects and related functionality that are used in many locations in an application. Connect image, text, image list and text list definitions to the interface of the widget definition to separate text or image content from widget design and to increase reusability.

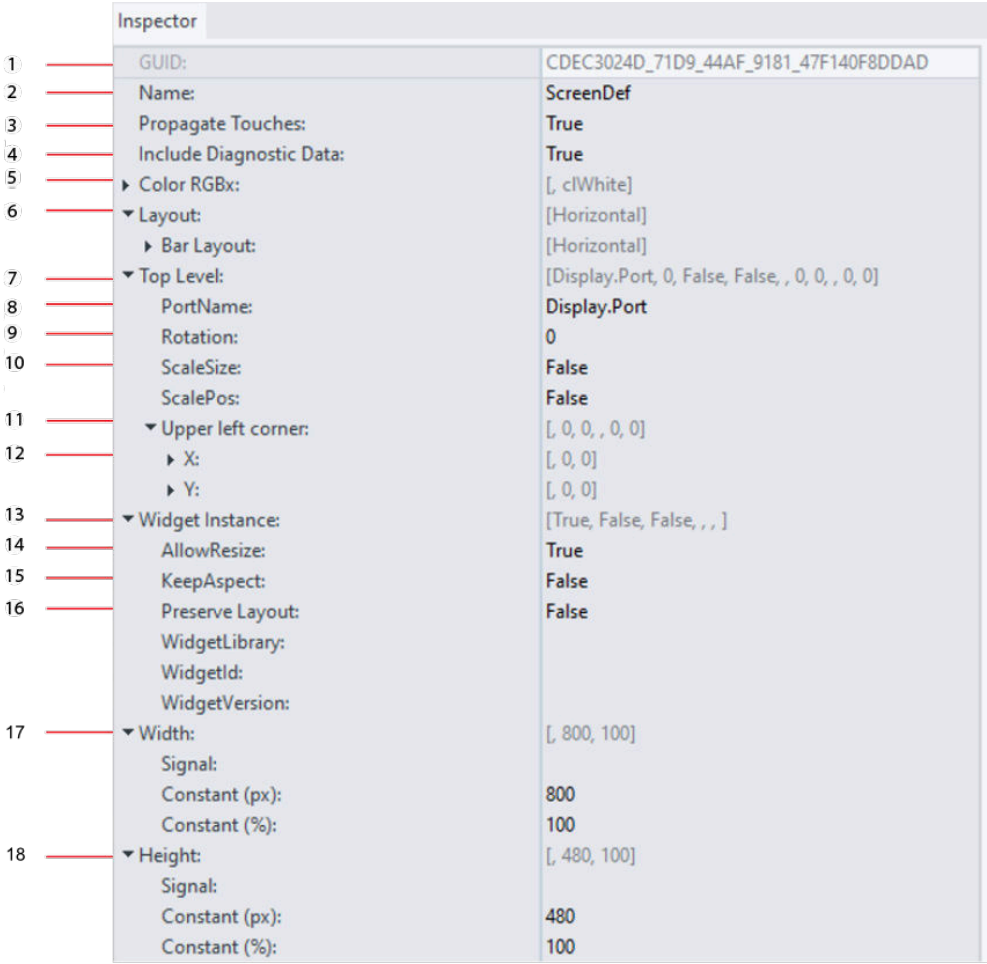
By exporting a widget definition, it may be reused in other applications. (See *About Exporting and Importing Screen Definitions* on page 517).

In theory, the same screen definition can be used both by a **Show Screen** component and as a widget, but there are some differences between these two kinds of usage. For instance, data types **Image** and **Text** can only be assigned to a screen definition being used as a widget and it is not possible to connect signals of these data types to an input bus of a **Show Screen** component. Some screen definition properties are only relevant in one of the two possible uses as described in *Screen Definition Properties* on page 488.

Data types **Text** and **STRING** are not the same. **Text** represents text definitions in Vector-Based Screen Editor and is not available in other parts of PLUS+1® GUIDE. Its purpose is to allow a widget to display different texts for different instances. **STRING** represents strings and is available throughout PLUS+1® GUIDE, see *STRING Types* on page 174.

Screen Editors

Screen Definition Properties



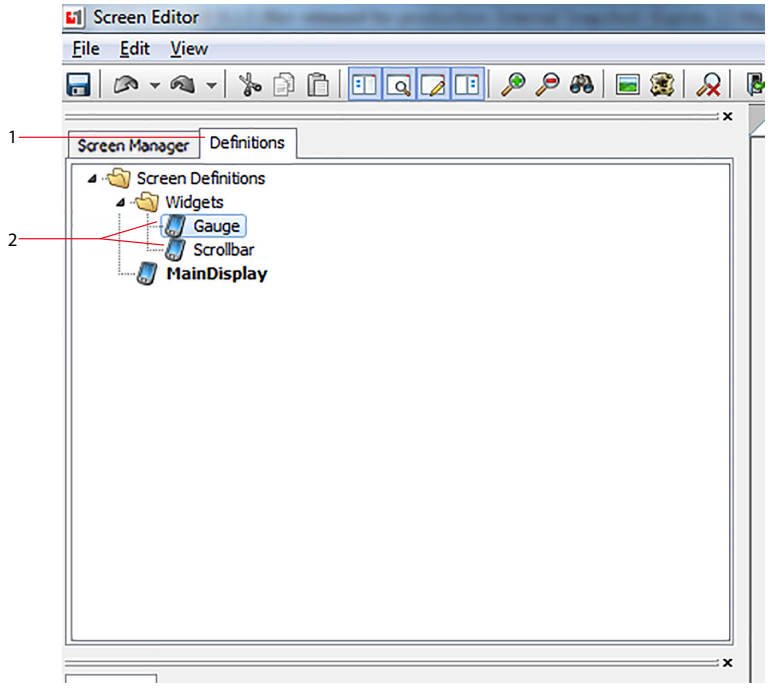
Item	Description
1	Internal unique id of the screen definition. Read-only.
2	Screen definition name.
3	Controls the behavior of touches within the screen definition. • If **False**, no touches will be detected in screen definitions underneath this screen definition • If **True**, touches may be detected in screen definitions underneath this screen definition. See *Touch Propagation* on page 524 for more details about this property. (Only available if the HWD supports touch.)
4	If **True**, Diagnostic and Non-volatile parameters of POUs contained in the screen definition will be included in the diagnostic data. (Only applicable when using the **Show Named Screen** component).
5	Background color of the screen definition. Possible values include a set of predefined colors: **Transparent** which will set the color to fully transparent, and **Custom Color…** which will open a dialog where the color can be freely defined. Double-clicking the will open the custom color dialog with the current color preselected. This may be useful for finding out the RGB code of a certain predefined color.
6	Layout management settings for this screen definition. See *Screen Definition Layout* on page 492.
7	Top level branch contains attributes that are relevant only when the screen definition is used as a top level screen definition. These attributes have no effect on widget instances.
8	The display on which to show this screen definition.

Screen Editors

Item	Description
9	Rotation is used when the display unit will be mounted at a rectangular rotation. (A widget instance will have the same rectangular rotation as its parent screen definition).
10	If scale size is true and the size of the display changes or the screen definition is imported to a project with a display of different size, the width and height of the screen definition will automatically be scaled. (Widget instances are screen objects, any scaling is applied separately for each widget instance. See *Manual Layout* on page 494, for **ScaleSize** description of screen objects in **Inspector** tab).
11	The constant value of **Upper left corner** will be scaled if the size of the display changes or the screen definition is imported to a project with a display of different size. (Widget instances are screen objects, any scaling is applied separately for each widget instance. See *Manual Layout* on page 494, for **ScalePosition** description of screen objects in **Inspector** tab).
12	X, Y coordinates of the upper left corner of the screen definition on the display. (A widget instance will be displayed on the same display as its parent screen definition.) (The insertion point of a widget instance is determined in its parent screen definition.)
13	Widget instance branch contains relevant attributes when the screen definition is used as a widget instance. These properties have no effect on top level screen definitions.
14	If **True**, it will be possible for widget instances of the screen definition to be resized. (The size of a top level screen definition is determined by its width and height properties.)
15	If **True**, the aspect ratio of the automatic size of the widget instances will be preserved. (Not applicable on top level screen definitions).
16	If **True**, and at least one of the screen objects inside this screen definition is using a layout manager, widget instances can be resized only if doing so does not violate the layout. For example, if **Preserve Layout** is true and there are no objects with flexible width, it will not be possible to change the width of the widget instances. (Not applicable on top level screen definitions).
17	Screen definition width. Constant for widget definitions, this defined width will be used as the default width of its instances, unless one or more of the screen objects are using layout manager. If layout manager is used, this defined width will only be applied when editing the widget definition.
18	Screen definition height. For widget definitions, this defined height will be used as the default height of its instances, unless one or more of the screen objects are using layout manager. If layout manager is used, this defined height will only be applied when editing the widget definition.

Screen Editors

Using Widgets

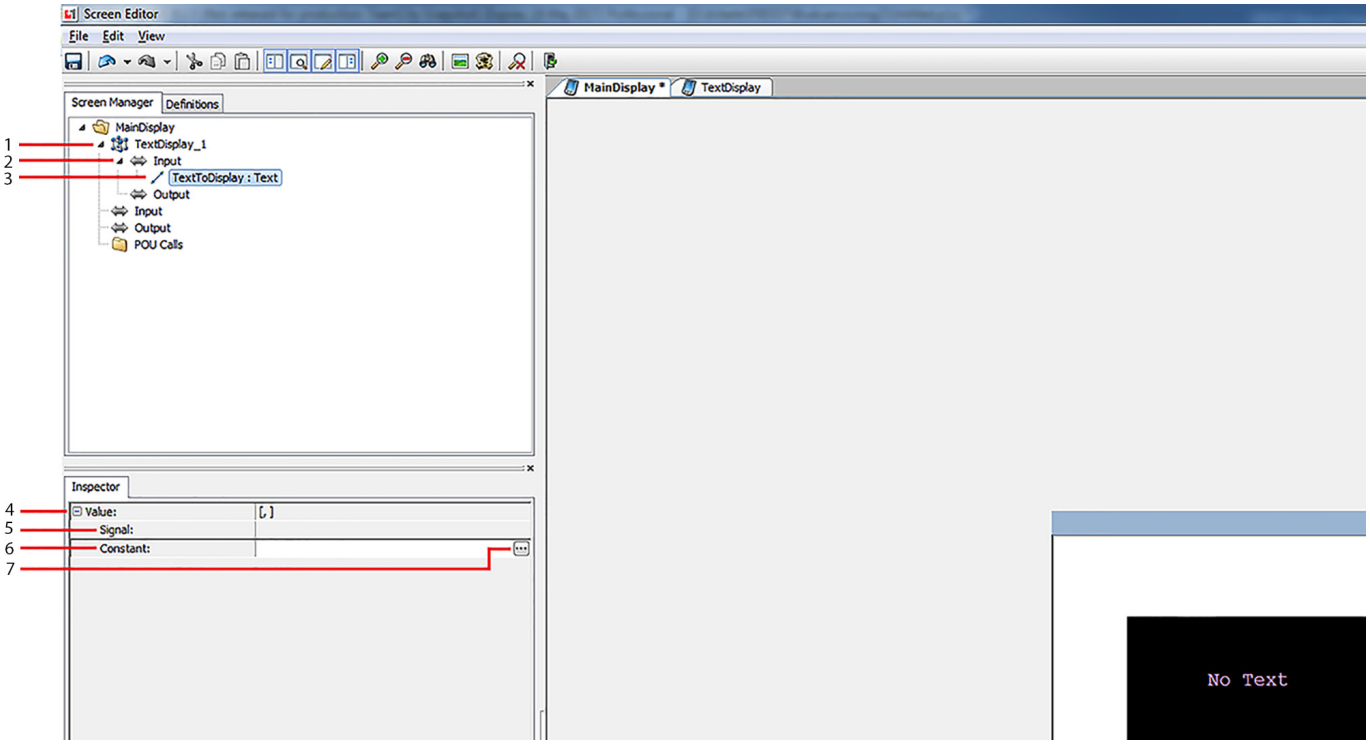


Adding a widget

Item	Description
1	The **Definitions** tab contains all screen definitions in a PLUS+1® GUIDE projects.
2	To add a widget, drag a screen definition from the **Screen Definitions** tree to the **Design Area**. Any screen definition can be added as a widget as long as it is not added to itself or contains itself as a widget.

The signal interface of a widget (widget interface) is accessed in the Screen Manager. Constant values in the widget interface will be reflected in the Design Area.

Screen Editors



Screen manager

Item	Description
1	Widgets are listed in the **Screen Manager** among other screen objects in the same screen definition. Open the context menu at a widget node and select **Edit Definition** to go to the widget screen definition.
2	The interface of a widget is available in the **Screen Manager**.
3	Select an interface signal to display it in the Inspector.
4	The selected interface signal, if any, is displayed in the **Inspector**.
5	Assign a signal. Available signals originate from the screen definition interface, other widgets, and POU calls.
6	Assign a constant value.
7	If the data type of the signal is **Color**, **Image**, or **Text**, this button opens a dialog where a value can be selected.

When a widget is added to the **Design Area**, it will automatically assume its preferred size. The **Autoresize** command, available in the context menu, is another way to make the widget assume its preferred size. If a widget has text or image inputs, the preferred size of the widget may change if those inputs are modified. The autoresize command will then resize the widget to its new preferred size. See *Layout* on page 491 for more information on how the widget is resized and how its subcomponents are sized and positioned.

Layout

Screen objects can be arranged manually or by a layout manager.

The **Layout Manager** property in the Inspector determines whether to use the layout manager specified by the parent screen definition (**Parent**) or not to use a layout manager (**None**). Depending on this setting, different sub-branches will be enabled in the inspector. See chapters *Layout Manager* on page 492 and *Manual Layout* on page 494, respectively.

Property **KeepAspect** is available regardless of the value of **Layout Manager**.

Screen Definition Layout

Screen definitions have a set of layout attributes affecting how they will behave when they are being used as widgets. They will have no effect when a screen definition is being referenced by a Show Screen component.

Each screen definition has a configurable layout manager. It will calculate the size and position of screen objects that are using it when the screen definition is resized.

When the Autoresize command is invoked for a widget, the size of the widget will be set to its preferred size. The preferred size will be determined by the layout manager of the widget if the widget contains at least one screen object that is using the layout manager (see *Autoresize* on page 493). Otherwise, it will be set to the design-time size of the screen definition.

There is one available layout manager called Bar Layout. It has a property called **Orientation** that configures it to be either **Horizontal** or **Vertical**. Bar Layout is described in detail in *Bar Layout* on page 492.

Screen Definition Layout Properties

Property	Description
AllowResize	Specifies whether a widget of this screen definition will be resizable.
KeepAspect	If true, the aspect ratio of this autoresized screen definition will be preserved.
Preserve Layout	A widget will only be resizable if its layout rules can be preserved. For example, if the screen definition only contains fixed-size objects, then there is only one size that fulfills the layout rules. This property has no effect if there are no objects using the layout manager.

Layout Manager

Screen objects are positioned and resized automatically according to a set of rules specified for each component that is using the layout manager. Use property **Layout Manager** to enable layout manager (see chapter *Layout* on page 491).

Bar Layout

Bar Layout will treat the screen definition as a one-dimensional bar. The property **Orientation** configures it to be either **Horizontal** or **Vertical**.

If the orientation is **Horizontal**, objects can be aligned in one of three **Alignment Zones**; Left, Center,and Right. Each zone has an ordering that specifies in which order the objects will be placed, this can be edited by drag-and-drop or the **Order** property of a screen object. A property called **Flexible Width** will allow an object in the Left or Right zone to be stretched or shrunk as the screen definition is resized. Objects also have a **Vertical Anchor** that specifies how the object is anchored vertically. Possible values are Top, Center, Bottom, and Top+Bottom. If Top+Bottom is selected, the object may be stretched and shrunk as the screen definitions resized. Finally, a property called Padding will add whitespace around an object. This whitespace will be included in the required size of an object.

Vertical orientation is similar to **Horizontal**, but objects are arranged on a vertical bar instead of a horizontal. There are three **Alignment Zones**: Top, Center, and Bottom. The **Order** property determines an object's order within its alignment zone. **Flexible Height** allows an object in the Top or Bottom zone to be stretched or shrunk in the vertical dimension. **Horizontal Anchor** specifies how to anchor the object in the horizontal dimension; possible values are Left, Center, Top, Left+Right, where the final value allows the object to be stretched or shrunk in the horizontal dimension. Padding has the same effect when orientation is vertical.

Lines

Lines are handled somewhat differently since other screen objects have an insertion point and a size while a line has a begin point and an end point.

Lines handled by layout management are always horizontal if the orientation of the layout manager is horizontal and vertical otherwise. Property **Length** defines the horizontal (vertical) extension of the line unless Flexible Width (height) is set. This corresponds to the width (height) of another screen object. The

Width of the line corresponds to the height (width) of another screen object, although the begin point of a line is at its vertical (horizontal) mid-point, not in the upper left corner. If the line object is anchored both at the top and bottom (left and right) edges, the **Width** property may be overridden. Lines have no **Keep Aspect** property.

Autoresize

When a widget is autoresized, all objects (including nested widgets) using its layout manager will be autoresized. Any padding will be added to the size of each object. All objects will be placed next to each other in their respective alignment zone, the size of the widget will be set to the smallest size that contains all objects using its layout manager. If there are objects in the central alignment zone the size of the other two zones will be equal, which may result in empty space inside the widget if one of the other zones is smaller than the other. If there is no central zone, the other two zones will not have any empty space between them. Screen objects not using a layout manager will be ignored when a widget is autoresized. A widget that is autoresized may become larger than the size of the display.

Resize

Resizing a widget may affect the size and position of screen objects using the layout manager. Size will only be affected if one of properties **Flexible Width**/**Flexible Height** is set or **Vertical Anchor**/ **Horizontal Anchor** is set to Top+Bottom/Left+Right; if **KeepAspect** is set the aspect ratio of the screen object will be preserved. The position will be affected in the following way if the orientation of the screen definition is Horizontal (Vertical): Objects in the Left (Top) alignment zone will be positioned in the defined order starting at the left (top) edge of the screen definition. The central alignment zone will be centered at the center of the screen definition. Objects in the Right (Bottom) alignment zone will be positioned from left to right (top to bottom) so that the right (bottom) edge of the last object will be positioned at the right (bottom) edge of the screen definition.

Resize of a widget may be affected by properties AllowResize, KeepAspect, and Preserve Layout in the screen definition. See *Screen Definition Layout* on page 492.

Layout Output

The position and size of a screen object after layout management has been performed can be obtained via four output signals. These properties can be used whenever the actual position or size of a screen object is needed, for example to position another screen object above an image to indicate scaling. Output signals are available in the inspector, in the branch **Layout/Bar Layout/Output**.

Bar Layout Properties

Property	Description
Alignment Zone	Determines in which zone the screen object will be located. Available choices are Left, Center, Right if the screen definition has horizontal layout and Top, Center, Bottom if the screen definition has vertical layout.
Flexible Width/Flexible Height	Makes the screen object stretchable and shrinkable. Note that **KeepAspect** may prevent an object from being stretched or shrunk; if **KeepAspect** is set then **Vertical Anchor** (**Horizontal Anchor**) must be set to Top+Bottom (Left+Right) for the object to be resizable.
Layout Order	The order from left to right (top to bottom), starting at zero, of a screen object in its **Alignment Zone**.
Length (lines only)	The horizontal (vertical) length of a line. (Note that the final length in the widget instance may be affected by other properties, other screen object, and widget size).
Vertical Anchor/Horizontal Anchor	Determines the position of a screen object in the vertical (horizontal) dimension. Possible values are Top, Center, Bottom, Top+Bottom (Left, Center, Right, Left+Right) where the last value will make the screen object stretchable or shrinkable. Note that **KeepAspect** may prevent an object from being stretched or shrunk; if **KeepAspect** is set then **Flexible Width** (**Flexible Height**) must also be set for the object to be resizable.

Bar Layout Properties (continued)

Property	Description
Padding	Add space around a screen object. Padding will increase the amount of space needed for a screen object. **Bottom, Left, Right, Top** determines the padding for each of the screen object's sides. If **UseAll** is set, then the value specified by **All** will be applied to each side of the screen object.
Output	Output position and size of the screen object after layout management. Line objects will output the begin and end coordinates.

Manual Layout

Screen objects are positioned in the Design Area using drag-and-drop or by modifying their manual layout properties in the **Inspector** or the **Edit Component** dialog. Use property **Layout Manager** to enable manual layout (see chapter *Layout* on page 491). Property **KeepAspect** will ensure that the width-to-height ratio is preserved when a screen object is resized in the Design Area, by modifying its properties or resizing a widget containing a screen object which also has **ScaleSize** set.

To arrange multiple screen objects at the same time, select the screen objects to arrange and open the context menu. Then expand sub-menu **Align/Distribute** and use any of the options to modify the layout of the selected objects.

When a widget is resized in design time, any screen objects having **ScaleSize** or **ScalePosition** set will be scaled or moved.

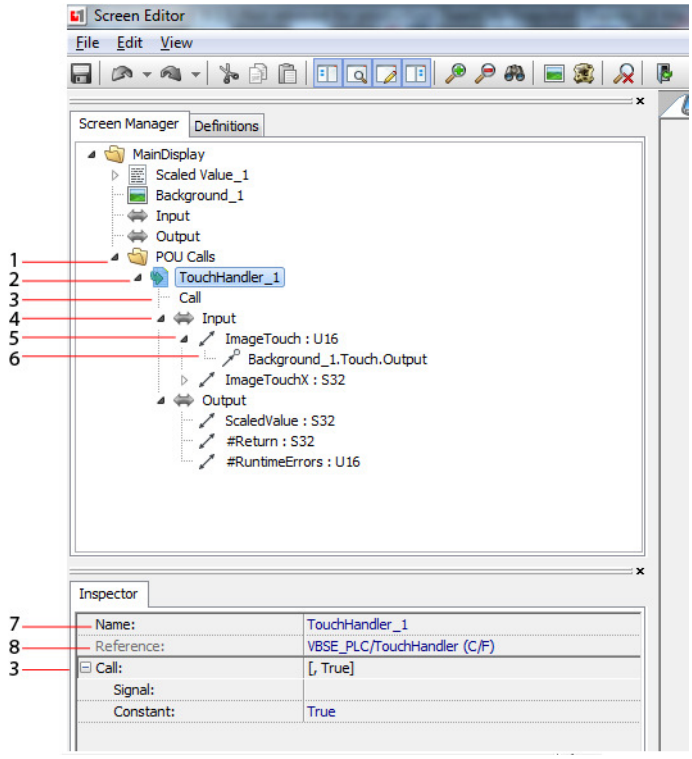
Manual Layout Properties

Property	Description
Insertion Point	X, Y coordinates of a screen object. By default, a screen object will be aligned so that the upper left corner is positioned at the insertion point. Objects may have an offset from the upper left corner using the Attachment Point property. With Attachment Point set to Default. **Images**, **Hardware Images**, and **Generic Viewports** may have an offset down to individual pixels from the upper left corner. This offset is defined at the referenced screen library image and the RotationPoint property of a hardware image or generic viewport. (Constant values can be given in pixels or in percentage of screen size)
Attachment Point	X, Y insertion point offset. The screen object will be inserted at and rotated around this point. 9 different Attachment Points are possible: Default, Center, Center Top, Right Top, Left Center, Right Center, Left Bottom, Center Bottom and Right Bottom. Default corresponds to the top left corner.
Width, Height (both properties are present)	Width and height of a screen object. These properties are constant for **Text**, **Text List**, **Image**, **Image List**, and **Widget**. Signal connections are allowed for **Touch Area**, **Hardware Image**, and **Generic Viewport**. (Constant values can be given in pixels or in percentage of screen size)
RotationPoint (hardware images and generic viewports only)	X, Y insertion point offset. The screen object will be inserted at and rotated around this point. 0, 0 corresponds to the top left corner.
Rotation	Clockwise rotation (degrees) of an **Image**, **Image List**, **Hardware Image**, or **Generic Viewport**.
Begin point (lines only)	(X, Y) coordinates of the begin point of the line. (Constant values can be given in pixels or in percentage of screen size)
End point (lines only)	(X, Y) coordinates of the end point of the line. (Constant values can be given in pixels or in percentage of screen size)
ScaleSize	If true, the size of the screen object will be resized proportionally when the size of the screen definition changes compared to its design-time size. Property KeepAspect will not be taken into consideration when ScaleSize is true.
ScalePosition	If true, the position of the screen object will be resized proportionally when the size of the screen definition changes compared to its design-time size.

Screen Editors

Call POU from screen

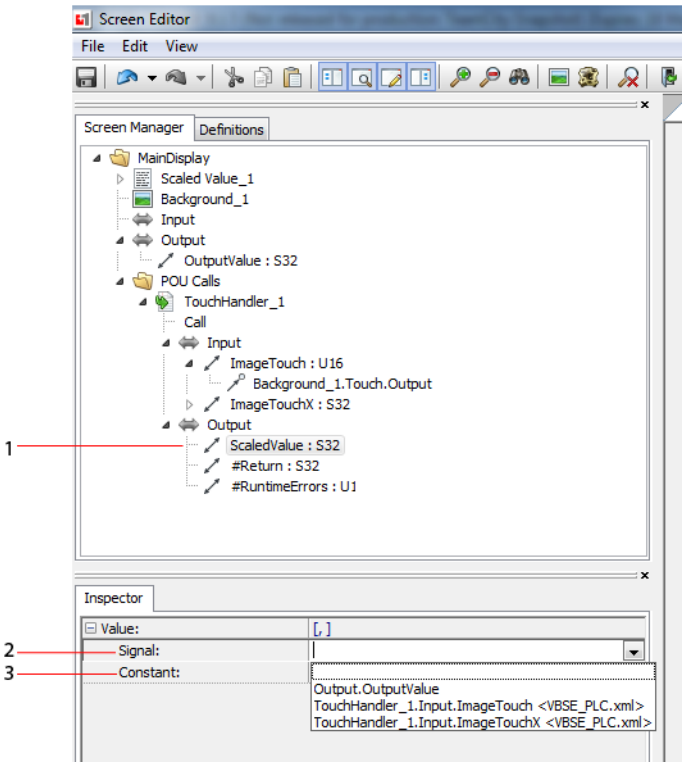
POU calls can be added in screen definitions. The interface of the POU connection will be updated automatically from the POU definition.



POU Calls

Item	Description
1	All **POU calls** in a screen definition are listed under the **POU Calls** branch. Open the context menu to add a call to a new POU or to add a call to an already existing POU. All POUs are listed in the PLC Units branch in the **Project Manager**.
2	Items directly under the **POU Calls** branch represent **POU Calls**. Use the context menu to select which POU to call and to open the selected POU in the **PLC/C Editor**. It is also possible to rename or delete this POU call.
3	Control when to perform this POU call. Use the inspector to control POU execution with a constant value or a signal.
4	The interface of the POU is shown in the **Screen Manager**. It is not possible to edit the interface here; whenever the interface is edited in the PLC/C editor it will be updated here.
5	Expand a POU interface bus to show the name and equivalent PLUS+1® GUIDE data type of its inputs or outputs.
6	The screen object connected to this signal, if any, can be displayed by expanding the signal node.
7	The name of the POU call can be edited in the Inspector.
8	This read-only property shows information about the referenced POU.

Screen Editors



Assignment of POU interface variables

Item	Description
1	Select a POU interface variable to edit it in the **Inspector**.
2	Open the **Signal** combo box to select a signal. Signals from other POU calls, widgets, and the screen definition interface are available. To connect a POU interface variable to a screen object property, use the signal selection combo box at that screen object property in the **Inspector**.
3	Use the **Constant** property (obscured by the signal selection combo box in the image above) to set a constant value.

If a screen definition has an internal state, then a **POU function block** must be used. If a **POU function** is used, no internal state will be remembered.

Internal Connections

The tasks of maintaining the bus interface and its connections in widgets and POU calls, can be facilitated by using the functionality described in *Signal Assignment Table* on page 497, *Connect Bus* on page 497, *Add and Connect Bus* on page 499, *Configure Object Interface* on page 501, and *Invalid Connections* on page 507.

Screen Editors

Signal Assignment Table

The signal assignment table is used as part of the Connect Bus, Add and Connect Bus and Configure Object Interface functionality. To access it, open the **Configure Object Interface** dialog of a widget or POU call.

Source Signal	Direction	Data Type	Constant	Connect	Target Signal	Status
Input.Speed	Input	S32				
Output.Indicators.HazardLight	Output	S16				
Output.Indicators.LeftIndicator	Output	S16				
Output.Indicators.RightIndicator	Output	S16				
Output.Speed.Max	Output	S32				
Output.Speed.Unit	Output	STRING				
Output.Speed.Value	Output	S32				

Column	Description
Source Signal	Lists all signals in the widget or POU call. Buses are delimited with a dot.
Direction	Direction of the signal.
Data Type	Data type of the signal.
Constant	Edit the constant value of an input signal.
Connect	Check to connect the source signal to the signal in the Target Signal column.
Target Signal	Select a target signal to connect to the source signal.
Status	If the selected target signal is ineligible, this field will explain why. In some cases a hyperlink will appear; clicking that hyperlink will modify the target signal.

The direction of signals in the Input bus of a widget or POU call is Input; signals in the Output bus of a widget or POU call have direction Output. On the other hand, in the screen definition interface, signals in the Input bus have direction Output and signals in the Output bus have direction Input.

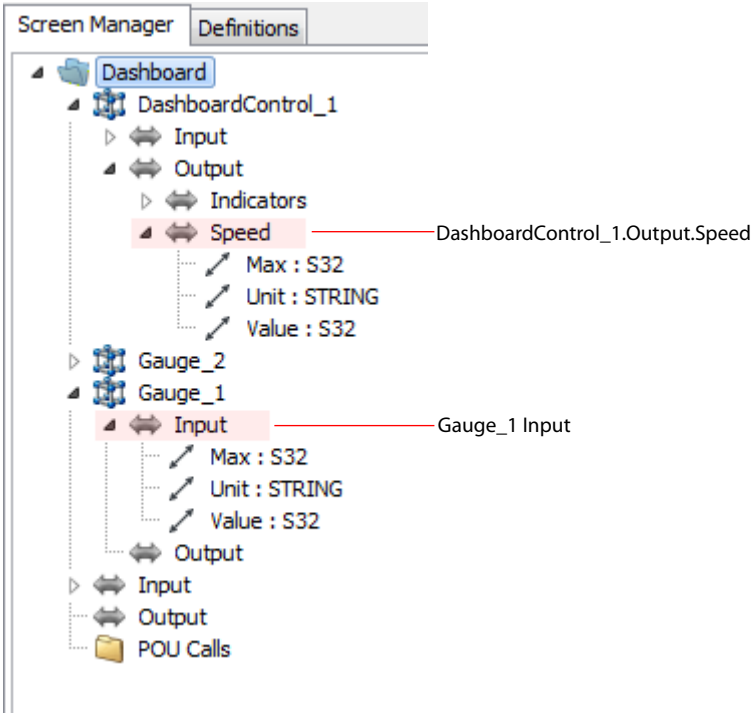
Connect Bus

Use this command to connect all signals in one bus to all matching signals (name and datatype) in another bus recursively. Connect Bus can be invoked from the context menu of any bus in a screen definition.

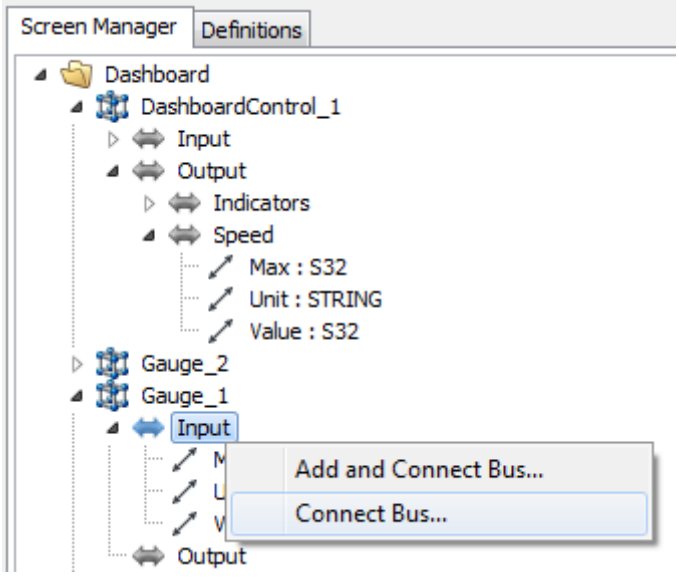
Example—Connect Bus

In the following example, the task is to connect the members of Gauge_1.Input to DashboardControl_1.Output.Speed.

Screen Editors

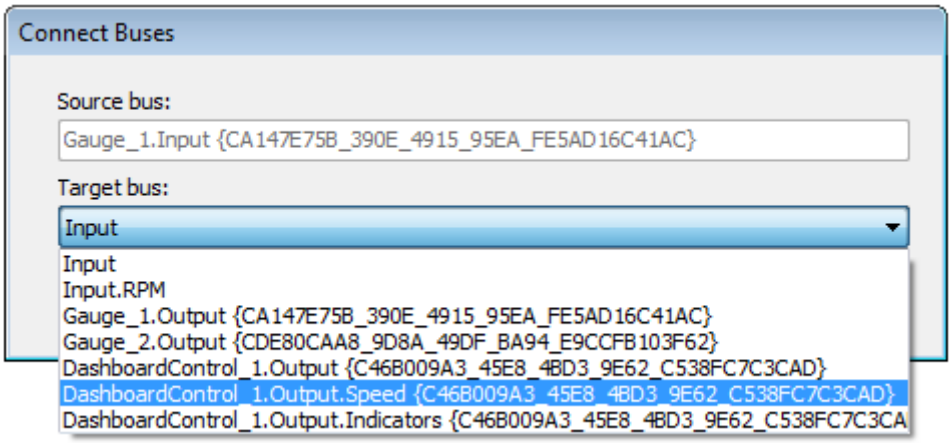


1. Right-click **Gauge_1.Input** to open the context menu and then click **Connect Bus....**

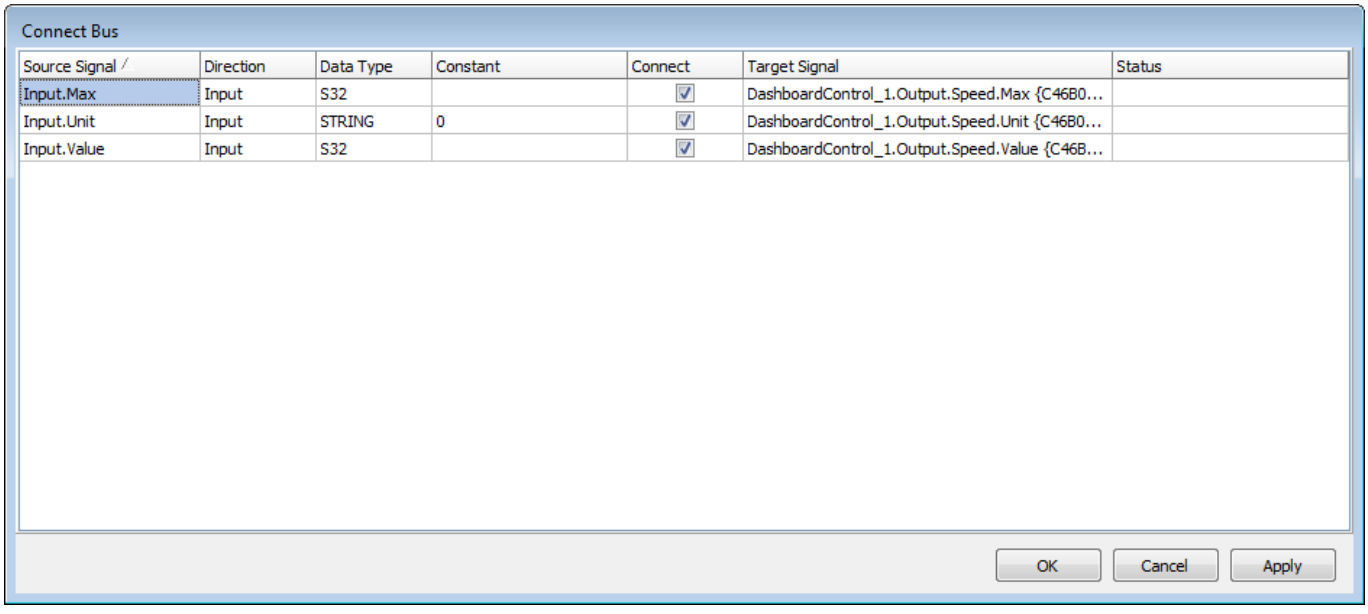


In the dialog that is opened, Source bus is the bus from which the command was invoked. The Target bus combobox contains all buses with matching direction from the screen definition interface, widget interfaces and POU call interfaces.

Screen Editors



2. Press **OK**.
   A final dialog will appear containing the signal assignment table of bus Gauge_1.Input.
3. Review and modify the result of the Connect Bus operation. Press **Apply** or **OK**.



The state of the signal assignment table will be applied.

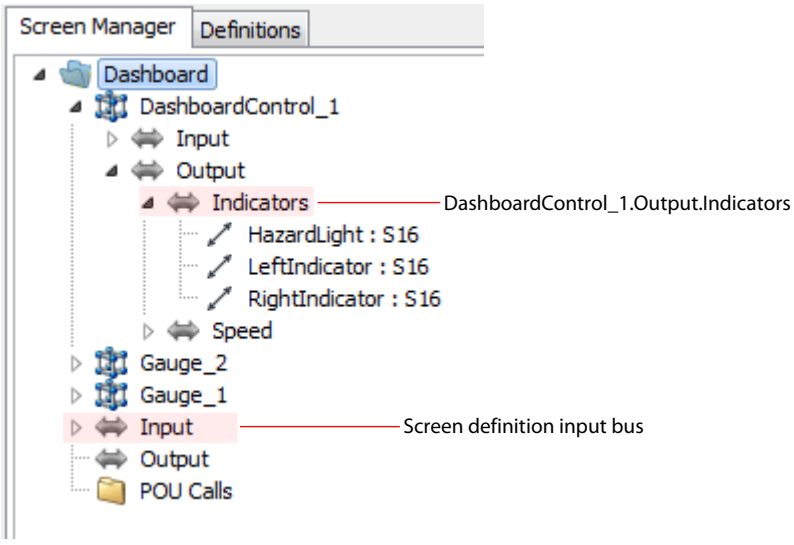
Add and Connect Bus

Add and Connect Bus command will add a copy of a bus as a sub-bus to another bus and connect the signals in the original bus to the signals in the new bus. The actions will be performed recursively. **Add and Connect Bus** can be invoked from the context menu of a widget or POU call bus.

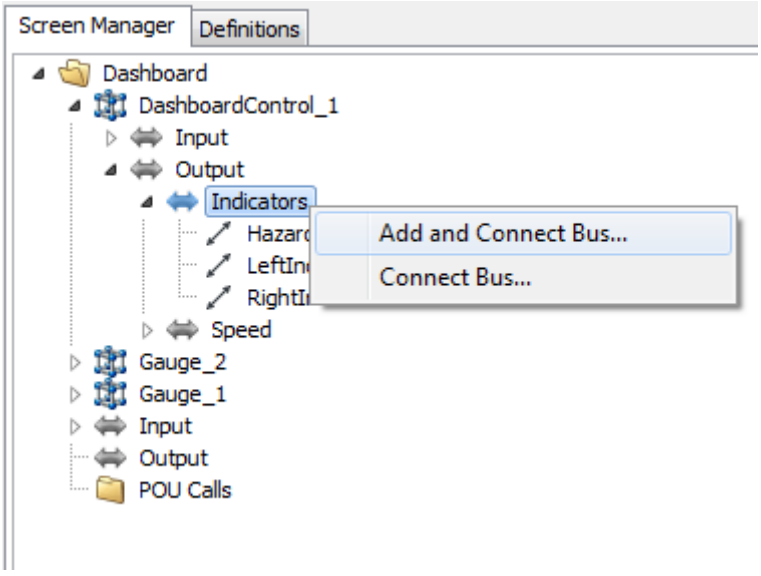
Example—Add and Connect Bus

In the following example, the task is to connect the members of DashboardControl_1.Output.Indicators to a new sub-bus in the screen definition Output bus.

Screen Editors

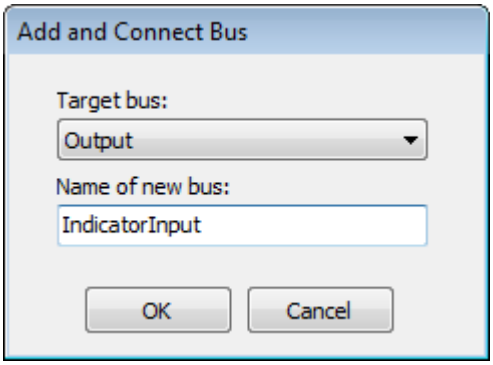


1. Right-click **DashboardControl_1.Output.Indicators** to open the context menu and then click **Add and Connect Bus….**



In the dialog that is opened, Target bus is the bus to which the copy of the source bus (in this example, DashboardControl_1.Output.Indicators) will be copied. The text input specifies the name of the copy. For the purpose of this example, the default text has been changed to IndicatorInput.

Screen Editors

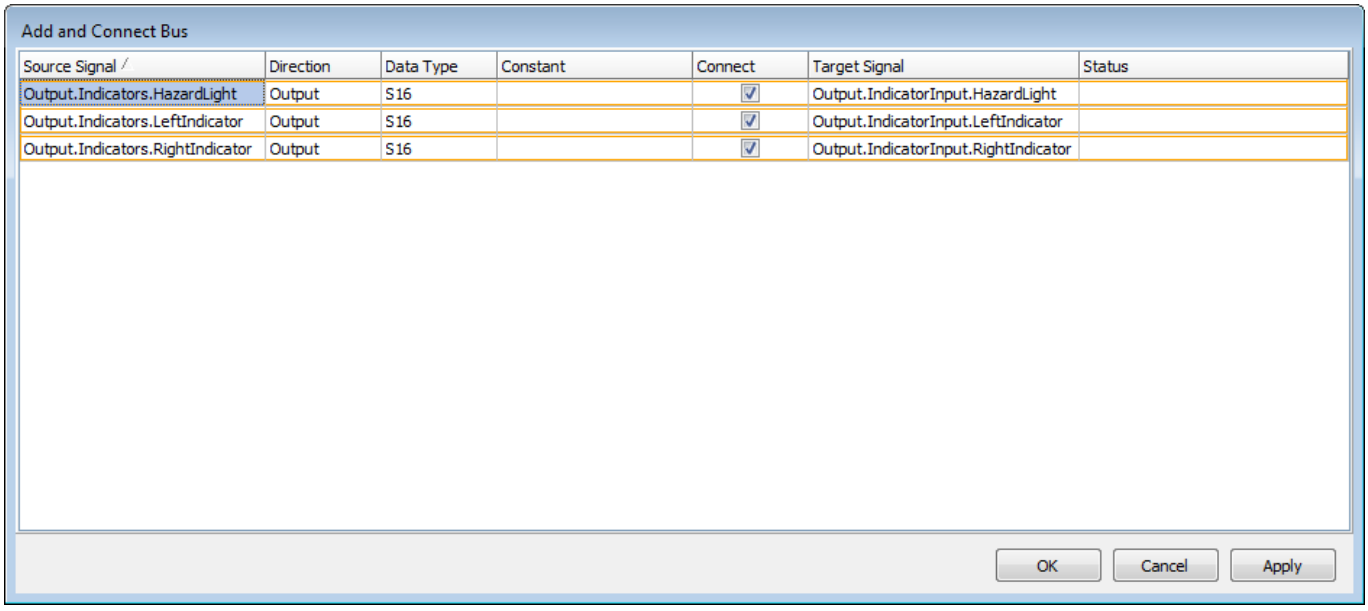


2. Press **OK**.

   A final dialog will appear containing the signal assignment table of DashboardControl_1.Output.Indicators.

3. Review and modify the result of the Add and Connect Bus operation. Press **Apply** or **OK**.

Pressing Apply or OK will apply the state of the signal assignment table. Lines in the table highlighted with an orange rectangle indicate that the target signal will be created when the state of the signal assignment table is applied. If the Connect check box is unchecked or another target signal is selected, the unchecked or unselected signal will not be created in the new bus.

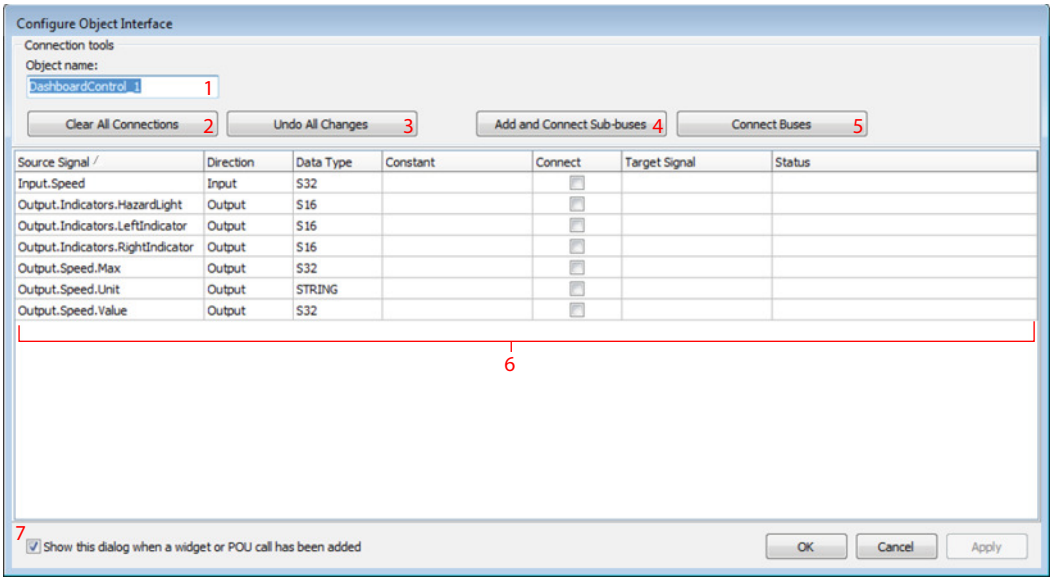


When a screen definition is edited, it is not possible to modify the bus interface of a widget definition or POU. Only the bus interface of the screen definition being edited is editable. As a consequence of this, the Add and Connect bus command may only target a bus in the screen definition interface.

Configure Object Interface

The **Configure Object Interface** dialog can be used to configure and connect a widget or POU call upon addition, or to manage the interface and connections of a widget or POU call at any time. It can be opened in two ways, either through the context menu of a widget or POU call, or automatically whenever a widget or POU call is added to a screen definition. The automatic opening can be controlled in the PLUS +1® GUIDE options.

Screen Editors



Item	Name	Description
1.	Object name:	Edits the name of this object.
2.	Clear All Connections	Will **Clear All Connections** in the signal assignment table of this dialog. No actual change will be performed to the object until **OK** or **Apply** is pressed.
3.	Undo All Changes	Will **Undo All Changes** made to the signal assignment table in this dialog since the last set of changes were applied.
4.	Add and Connect Sub-buses	Invokes the Add and Connect Bus command in a semi-automatic way. If non-empty, the Input bus of the object will be added and connected to the Input bus of a screen definition as a sub-bus, having the same name as the object. The Output bus of the object will be handled correspondingly. To use this function, the object name must not contain any other character than A..Z, a..z, _ and 0..9.
5.	Connect Buses	Invokes the Connect Bus command. When invoked in this way, it will be possible to select one of the sub-buses of the object as Source bus.
6.	Signal assignment table	The signal assignment table provides an overview of the object's interface and means to edit constants and connections. Any command performed by pressing any of the four upper buttons may be reflected in this table.
7.	Enable automatic control	Controls whether to show the **Configure Object Interface** dialog upon addition of a widget or POU call. Modifying this check box will modify the corresponding PLUS+1® GUIDE option.

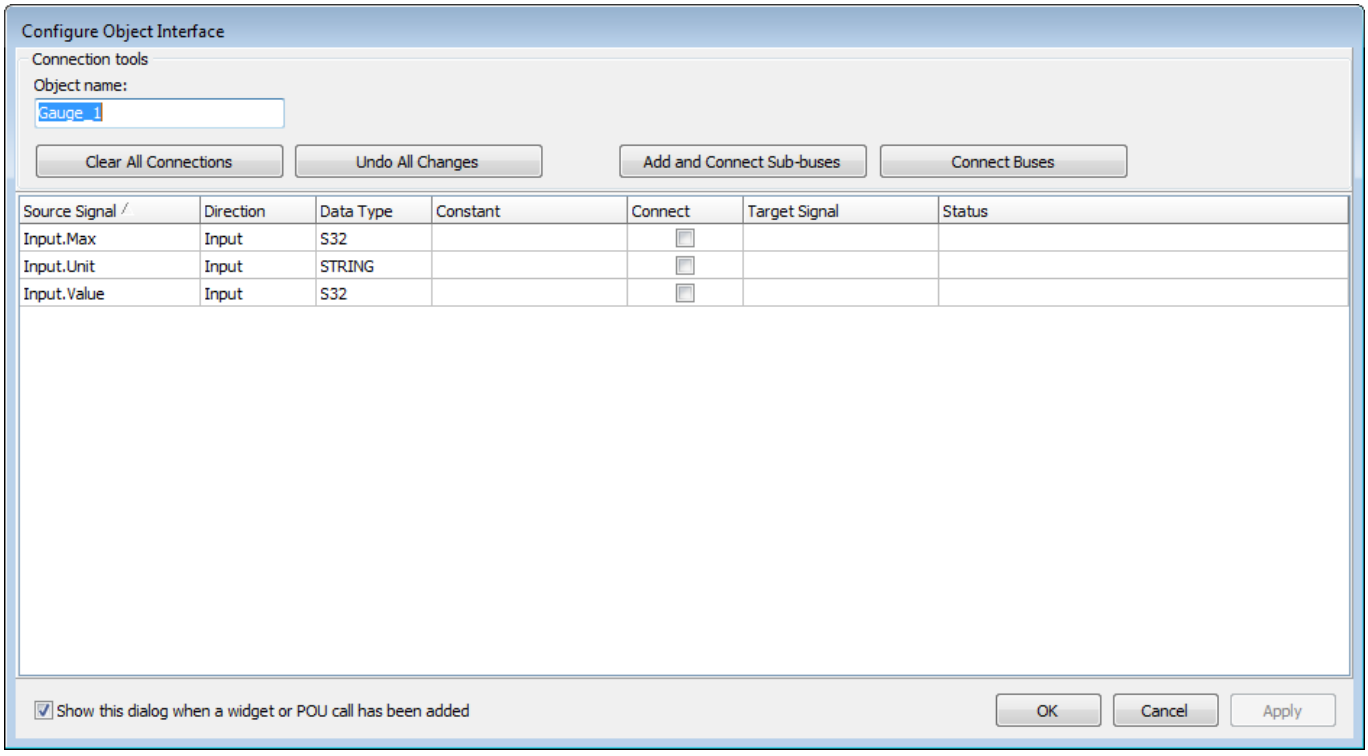
Example—Create New Screen Definition Input Bus

In the following example, the task is to add a widget with three inputs and create new screen definition input bus. Two of the inputs will be connected to the screen definition interface while the third input will get a constant value.

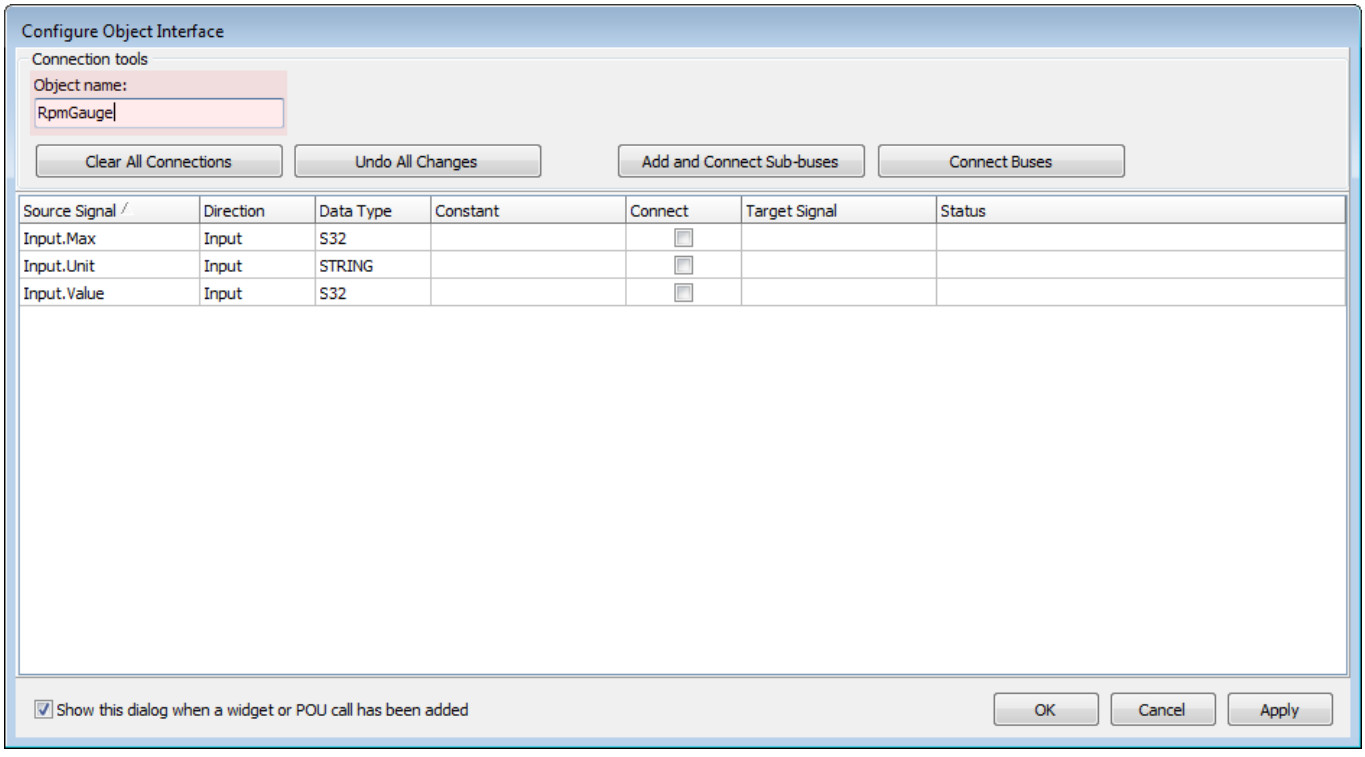
Screen Editors

1. Add widget Gauge to screen definition Dashboard.

   The **Object Interface Configuration** dialog will be opened automatically.



2. Change **Object name:** to RpmGauge (to give the widget a better name for the purpose of this example).

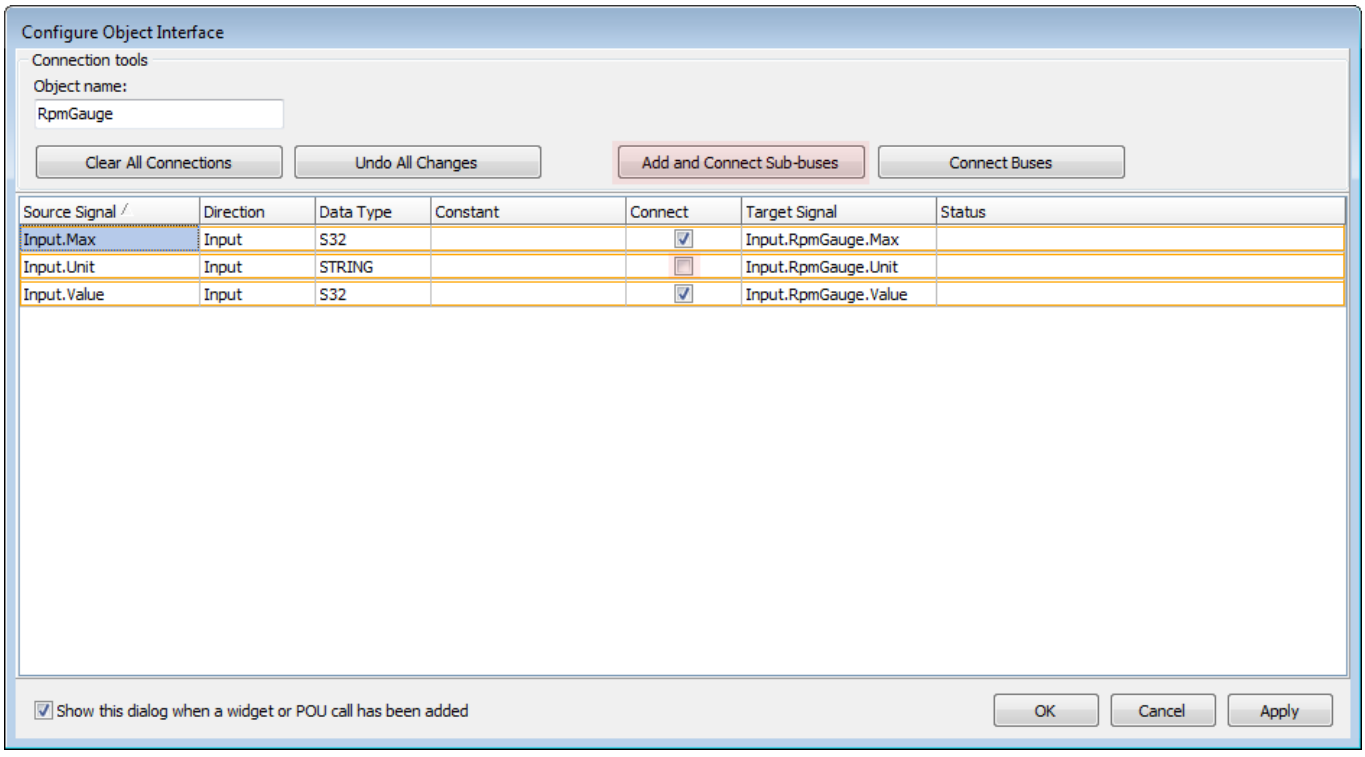


Screen Editors

3. Press **Add and Connect Sub-buses**.

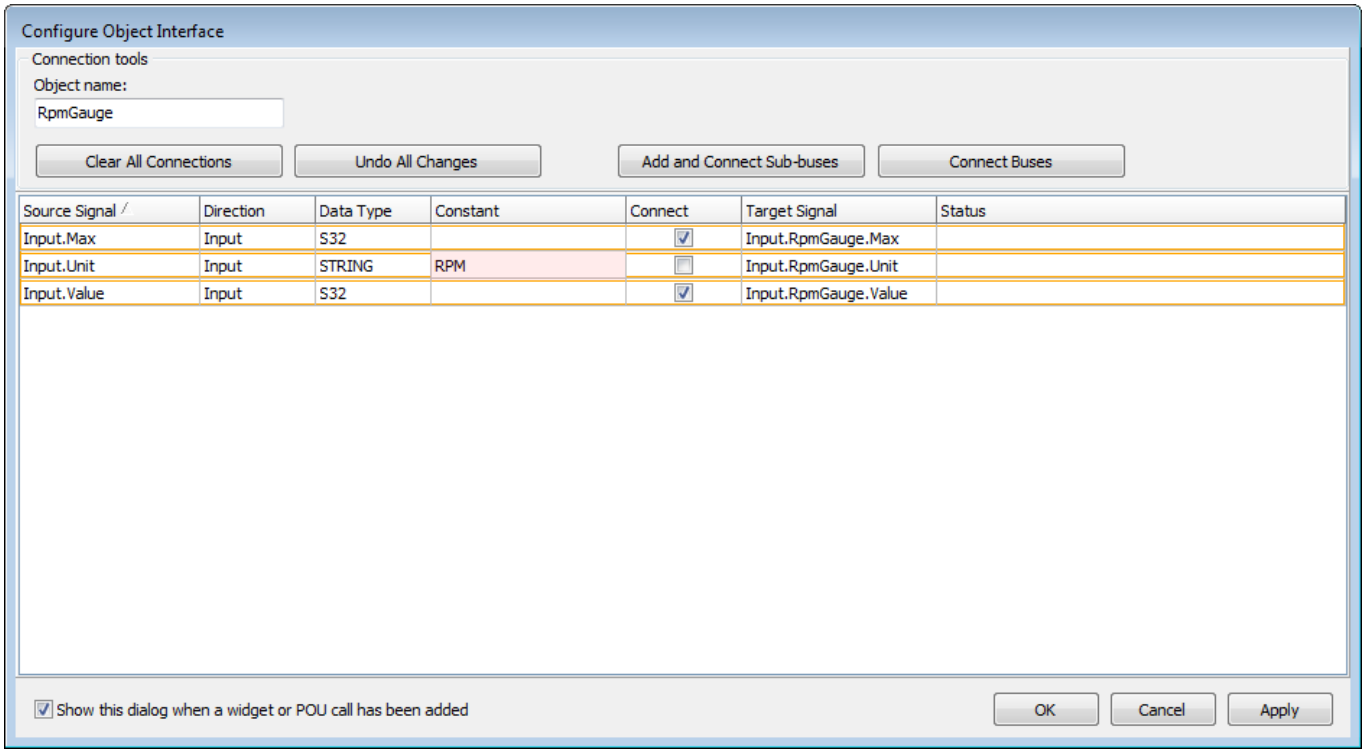
   The Connect check box of the signal called Unit is unchecked because it will not be added.

   Max and Value signals will be connected and a sub-bus in the screen definition interface will be created.



Screen Editors

4. Specify the STRING constant **RPM** for input Unit.



5. Press **OK**.

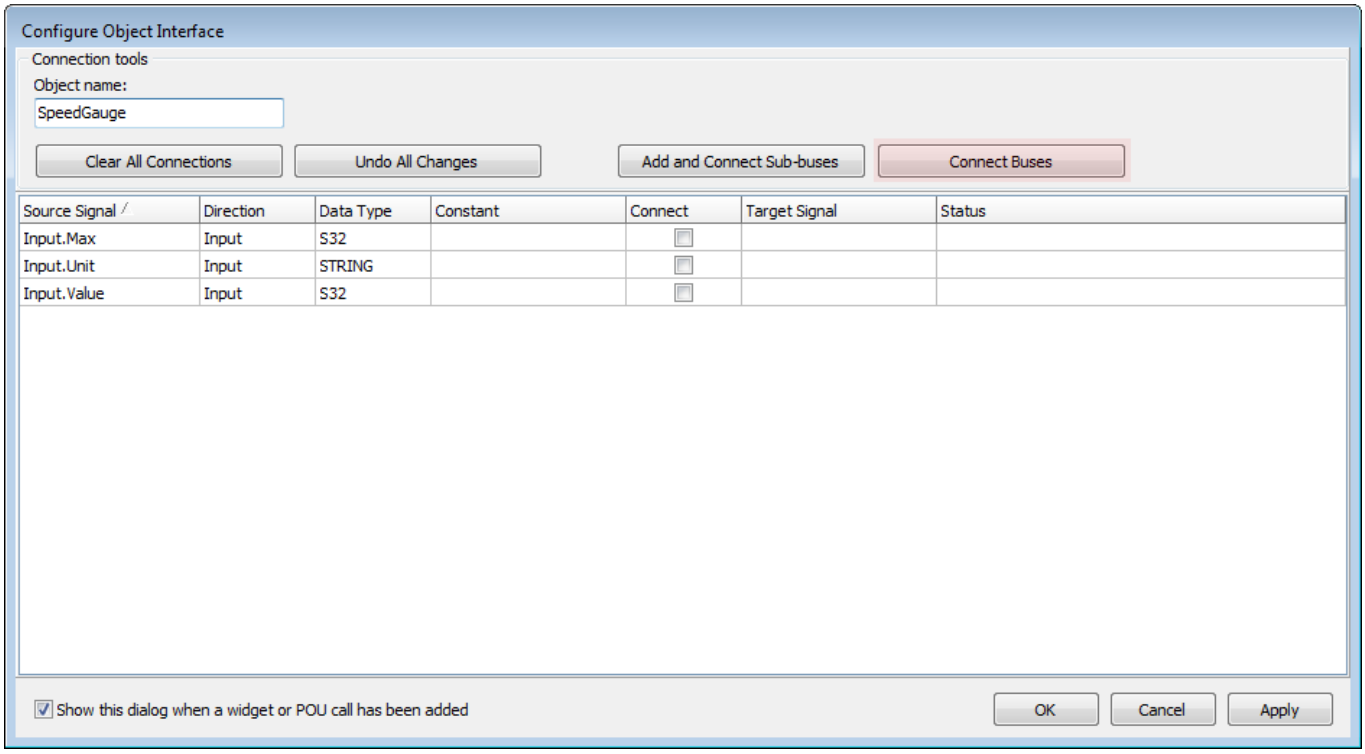
The bus interface of widget RpmGauge is configured and the input bus of the screen definition has got a sub-bus called RpmGauge with two members Max and Value.

Screen Editors

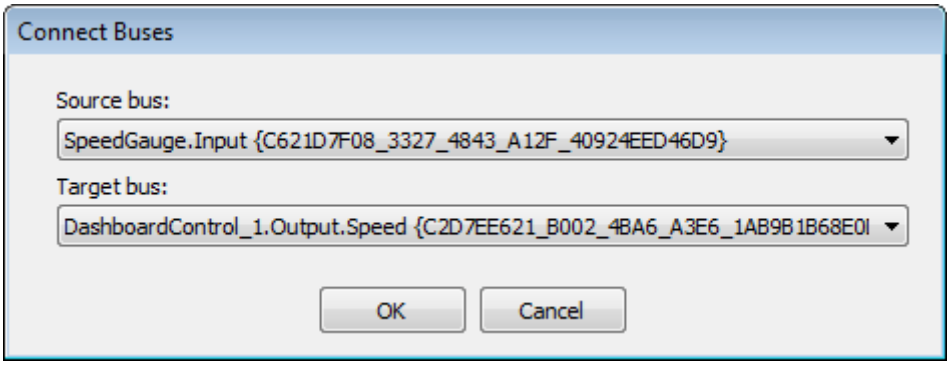
Example—Connect Widget to Another Widget

In the following example, the task is to add a widget and connect its input bus to an output bus of another widget.

1. Add widget Gauge to screen definition Dashboard.
   The **Object Interface Configuration** dialog will be opened automatically.
2. Change **Object name:** to SpeedGauge (to give the widget a better name for the purpose of this example).
3. Press **Apply** (to apply the new widget name to its existing bus structure names).
4. Press **Connect Buses** (to connect the input bus of the widget to an output bus of the other widget).



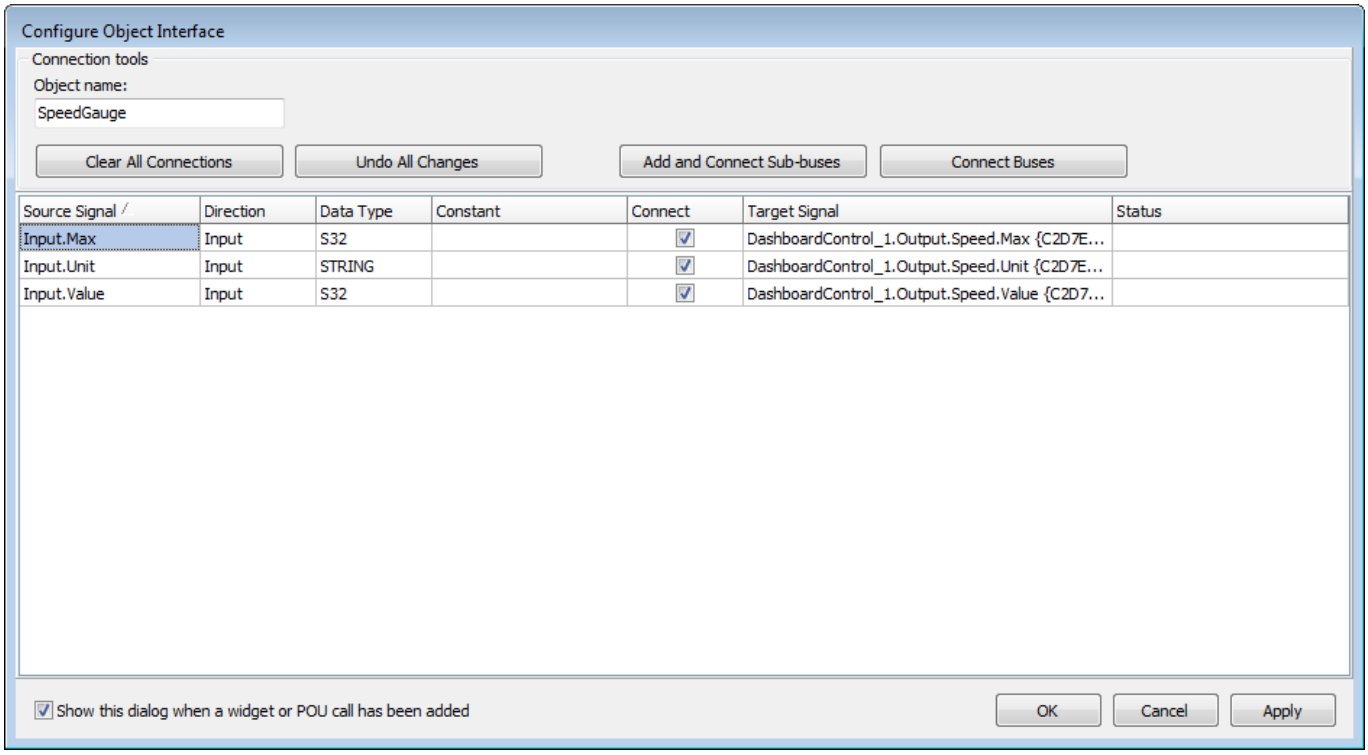
5. Select **Source bus:** SpeedGauge.Input and **Target bus:** DashboardControl_1.Output.Speed in the **Connect Buses** dialog.



6. Press **OK**.

The **Configure Object Interface** dialog is set up. Pressing **Apply**, the changes made are applied and the bus interface of SpeedGauge will be connected.

Screen Editors



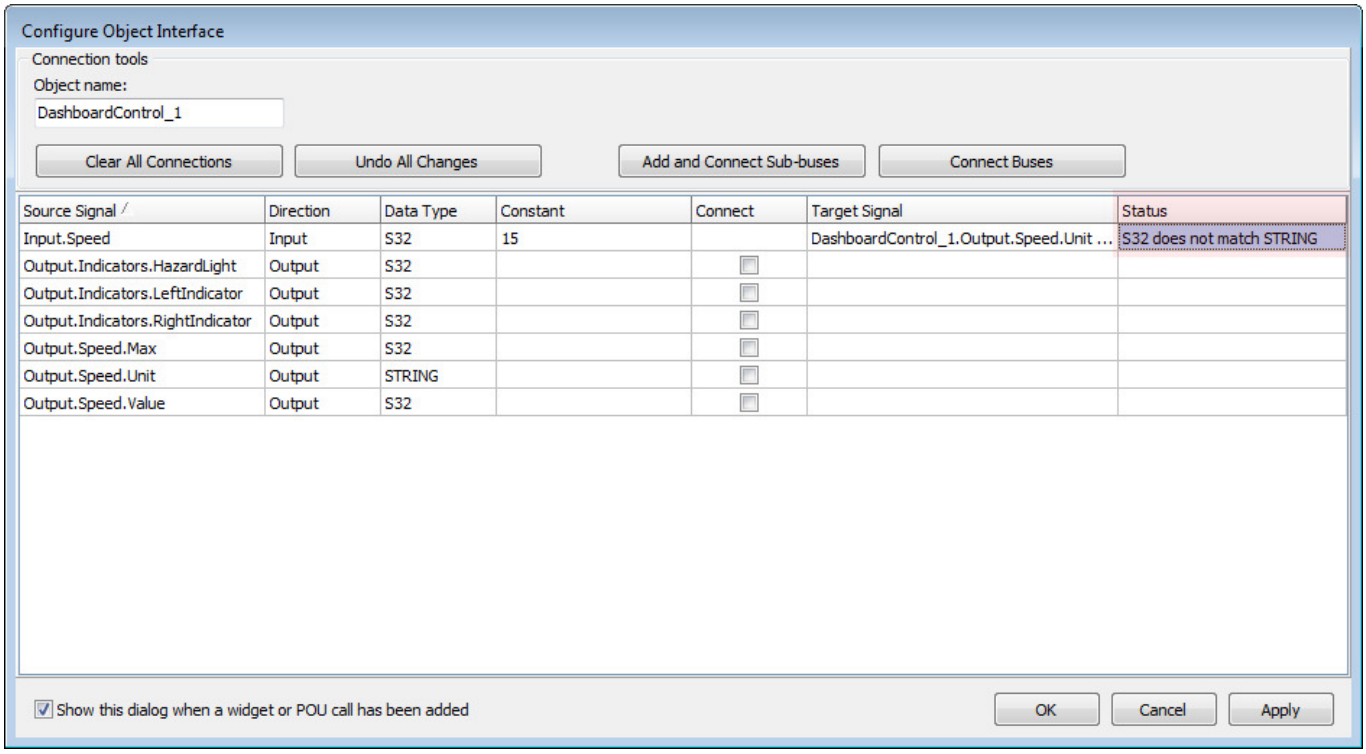
Invalid Connections

If the internal connection functionality results in an invalid connection, a message will appear under Status in the signal assignment table or an invalid connection dialog will appear.

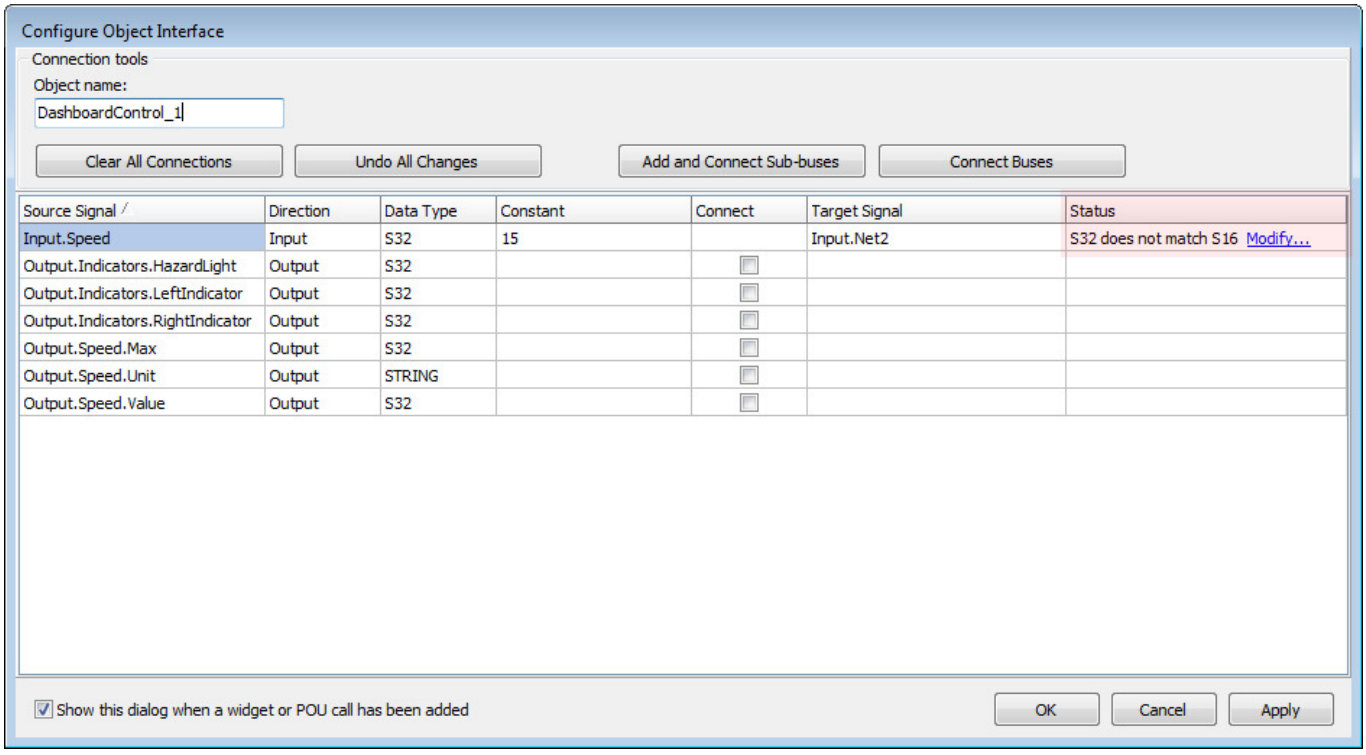
Data Type Mismatch

In the signal assignment table, if a Target Signal with incompatible Data Type is selected, an error message will be shown under Status and the Connect check box will not be visible and impossible to select.

Screen Editors



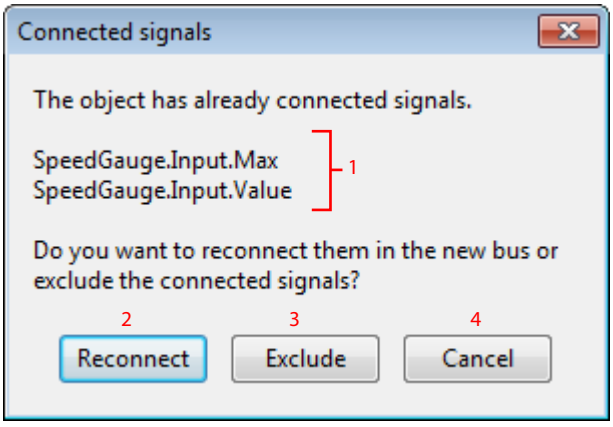
If the Target Signal is in the screen definition Input or Output bus, it is possible to change the Data Type of the screen definition signal. This is accomplished by clicking the **Modify…** hyperlink appearing under Status.



Screen Editors

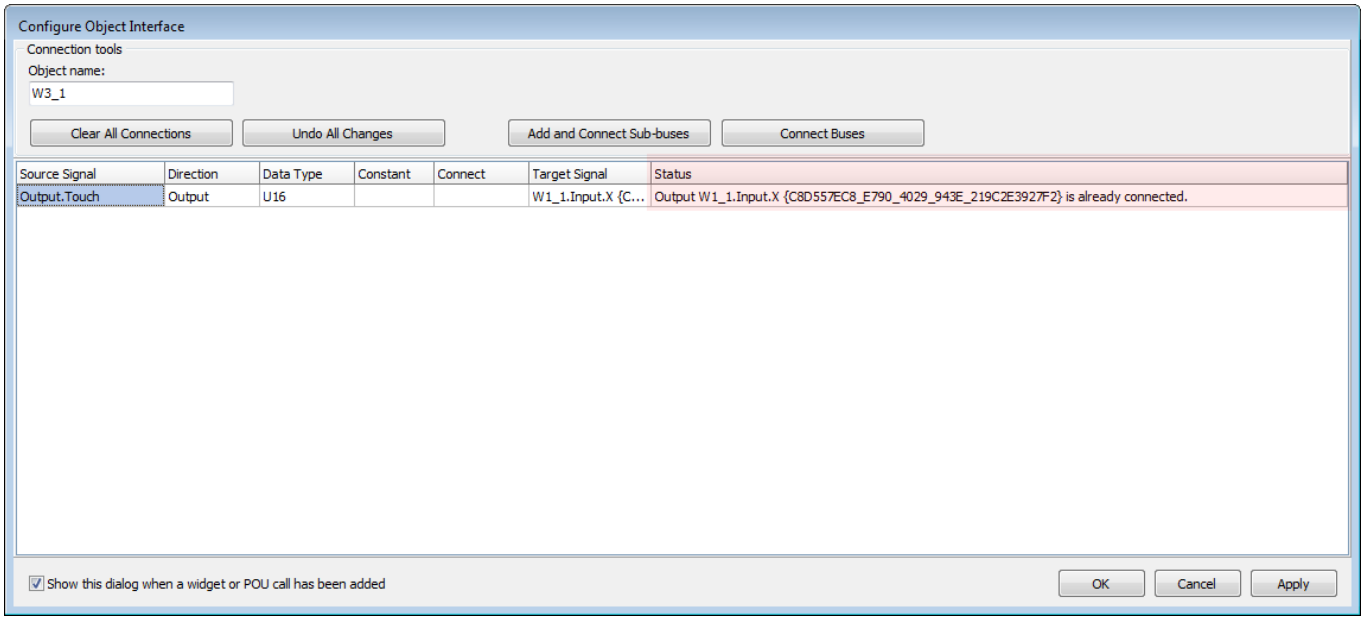
Connected Signals

A dialog is shown if a signal has connections that makes it ineligible for a Connect Bus or Add and Connect bus operation.



Item	Name	Description
1.	Signal list	All signals affected by the decision made in this dialog.
2.	Reconnect	Proceed with the operation and remove the previous connections of the signals in the list.
3.	Exclude	Proceed with the operation but do not include the signals in the list.
4.	Cancel	Cancel the entire operation.

If an input signal, connected to an output, is selected as a Target Signal for another output, an error message will be shown under Status and the Connect check box will not be visible and impossible to select.

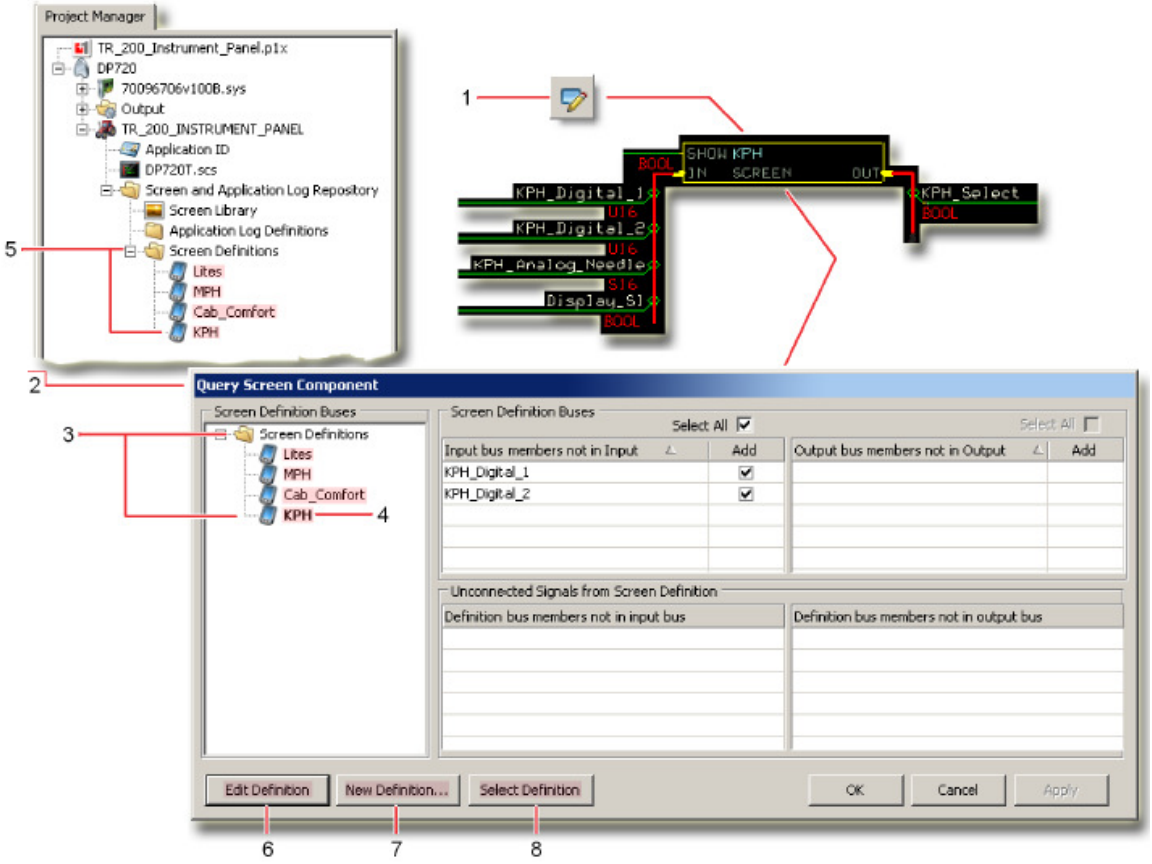


Screen Editors

About the Show Screen Component and the Query Screen Component Window

Use the **Query Screen Component** window to:

- Make application signals connected a **Show Screen** component's **IN** and **OUT** buses available for assignment within the **Screen Definition** instantiated by the **Show Screen** component.
- List the signals within the instantiated **Screen Definition** that are not connected to the application through its instancing **Show Screen** component's **IN** and **OUT** buses.
- Edit existing **Screen Definitions** and create new **Screen Definitions**.



1. Query a **Show Screen** component to open the **Query Screen Component** window.

   The **Show Screen** component in the preceding figure instantiates the **Screen Definition** for **Lites**.
2. **Query Screen Component** window:
   - Use to edit existing **Screen Definitions** and create new **Screen Definitions**.
   - Use to manage unconnected **Screen Definition** signals.
3. **Screen Definitions** lists all the **Screen Definitions** that are available in the application. The **Project Manager** tab shows the same list.
4. **Bold** text indicates the **Screen Definition** that the queried **Show Screen** component instantiates (**Lites** in the preceding figure).

   Click this **Screen Definition** to view information about unconnected signals between the **Show Screen** component and the **Screen Definition** that it instantiates.
5. **Screen Definitions** lists all the **Screen Definitions** that are available in the application. The **Query Screen** component window shows the same list.
6. **Edit Definition** button. Click to open the selected **Screen Definition** in the **Screen Editor** window.
7. **New Definition** button. Click to create a new **Screen Definition**.
8. **Select Definition** button. Click to instantiate a selected **Screen Definition** through a queried **Show Screen** component.

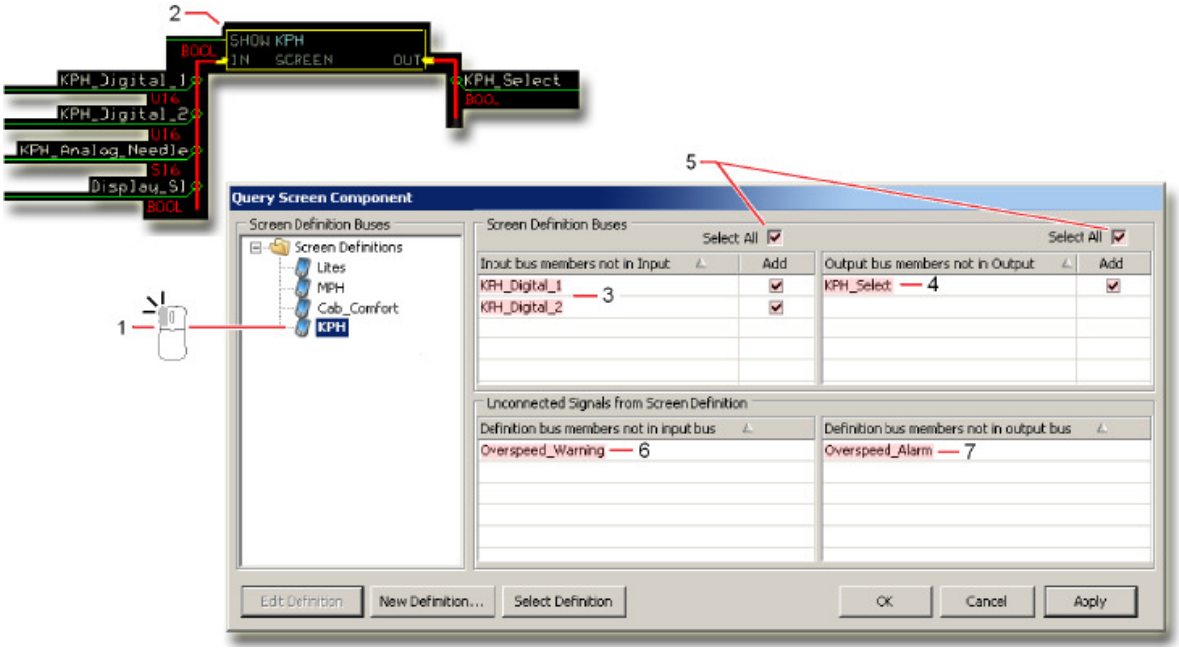
Screen Editors

About the Query Screen Component Window and Screen Definition Signals

Click a **Screen Definition** to open the **Query Screen Component** window.

- This window lists unconnected signals between the queried **Screen Definition** and its instancing **Show Screen** component.
- This window lists no signals if all the signals between the queried **Screen Definition** and its instancing **Show Screen** component are connected.

Not every signal listed in the **Query Screen Component** window necessarily needs a connection. You only have to connect the signals needed by the **Screen Definition** and its instancing **Show Screen** component.



1. **Screen Definitions** Lists all the **Screen Definitions** in your project. Click a **Screen Definition** to have the **Query Screen Component** window show information about:
   - Application signals connected to the **IN** and **OUT** bus of a **Show Screen** component have not been made available for assignment to assets inside the **Screen Definition** instantiated by the **Show Screen** component.
   - Signals inside the instantiated **Screen Definition** that have no connections out to the **IN** and **OUT** buses of instancing **Show Screen** component.
2. **Show Screen** component instantiates a **Screen Description**.
3. **Input bus members not in Input** lists application signals that:
   - Connect to the **IN** bus of the **Show Screen** component that instantiates the selected **Screen Definition**.
   - Have not been added to the **Input** signals within the **Screen Definition** that are available for assignment to screen assets.
4. **Output bus members not in Output**

   Lists application signals that:
   - Connect to the **OUT** bus of the **Show Screen** component that instantiates the selected **Screen Definition**.
   - Have not been added to the **Output** signals within the **Screen Definition** that are available for assignment to screen assets.
5. **Select All/Add** check boxes when enabled, these check boxes make signals available for assignment inside the **Screen Definition**.

Screen Editors

- **Select All**—adds all the signals to the signals that are available for assignment to screen assets.
- **Add**—adds the selected signals to the signals that available for assignment to screen assets.

  To enable the check boxes, connect the bus with the listed signals to a **Show Screen** component that instantiates the selected **Screen Definition**.

6. **Definition bus members not in input bus** lists **Screen Definition** signals that:
   - Are available as **Input** signals inside the selected **Screen Definition**.
   - Need a connection out to the application through signals routed to the **IN** bus of the **Show Screen** component that instantiates the selected **Screen Definition**.
7. **Definition bus members not in output bus** lists **Screen Definition** signals that:
   - Are available as **Output** signals inside the selected **Screen Definition**.
   - Need a connection out to the application through signals routed to the **OUT** bus of the **Show Screen** component that instantiates the selected **Screen Definition**.

Screen Editors

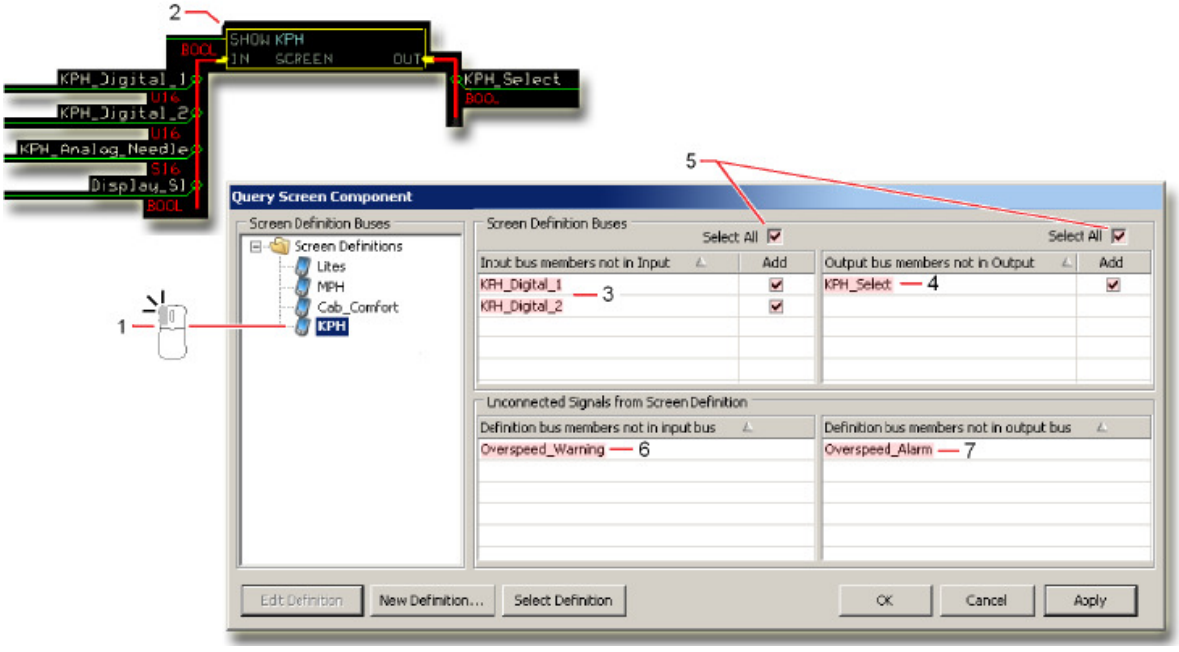
About the Query Screen Component Window and Screen Definition Buses

The following figure shows what the **Query Screen Component** window displays when you click a **Screen Definition** where:

- Not all the application signals connected to the **Show Screen** component's **IN** and **OUT** buses are available for assignment inside the instantiated **Screen Definition**.
- Not all signals inside the **Screen Definition** connect out to the application through the instancing **Show Screen** component's **IN** and **OUT** buses.

Not every signal listed in the **Query Screen Component** window necessarily needs a connection. You only have to connect the signals needed by the **Screen Definition** and its instancing **Show Screen** component.

Query Screen Component window—when not all signals are connected



1. This **Show Screen** component instantiates the **KPH Screen Description**.
2. Click to view:
   - Signals from the application that connect to the **Show Screen** component's **IN** and **OUT** buses but are not inside the Screen Definition.
   - Signals inside the Screen Definition that have no connection to the **Show Screen** component's **IN** and **OUT** buses.
3. The **Query Screen Component** window lists the **KPH_Digital_1** signals because they:
   - Connect to the **Show Screen** component's **IN** bus.
   - Are not available as Input signals inside the Screen Definition.
4. The **Query Screen Component** window lists the **KPH_Select** signal because it:
   - Connects to the **Show Screen** component's **OUT** bus.
   - Is not available as an Output signal inside the Screen Definition.
5. Check to select the signals that you want inside the Screen Definition.
6. Click this button to make the checked signals available for assignment inside the Screen Definition.
7. The **Query Screen Component** window lists the **Overspeed_Warning** signal because it:

Screen Editors

- Is available as an Input signal inside the Screen Definition.
- Needs a connection to the application through the **Show Screen** component's **IN** bus.

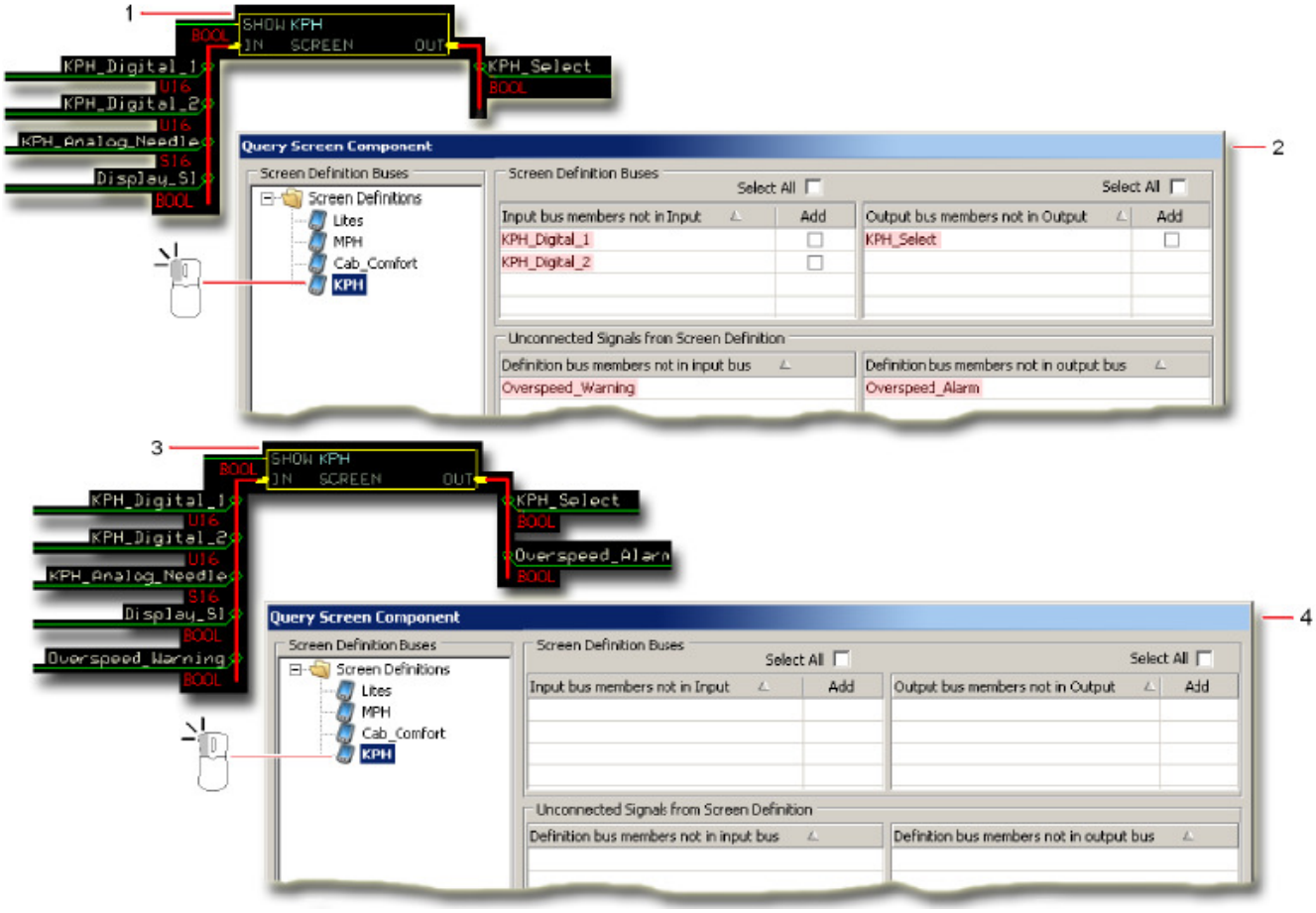
8. The **Query Screen Component** window lists the **Overspeed_Alarm** signal because it:

- Is available as an Output signal inside the Screen Definition.
- Needs a connection to the application through the **Show Screen** component's **OUT** bus.

Screen Editors

About the Query Screen Component Window and KPH Screen Definition

Query Screen Component window before and after you resolve all connection issues to a Screen Definition



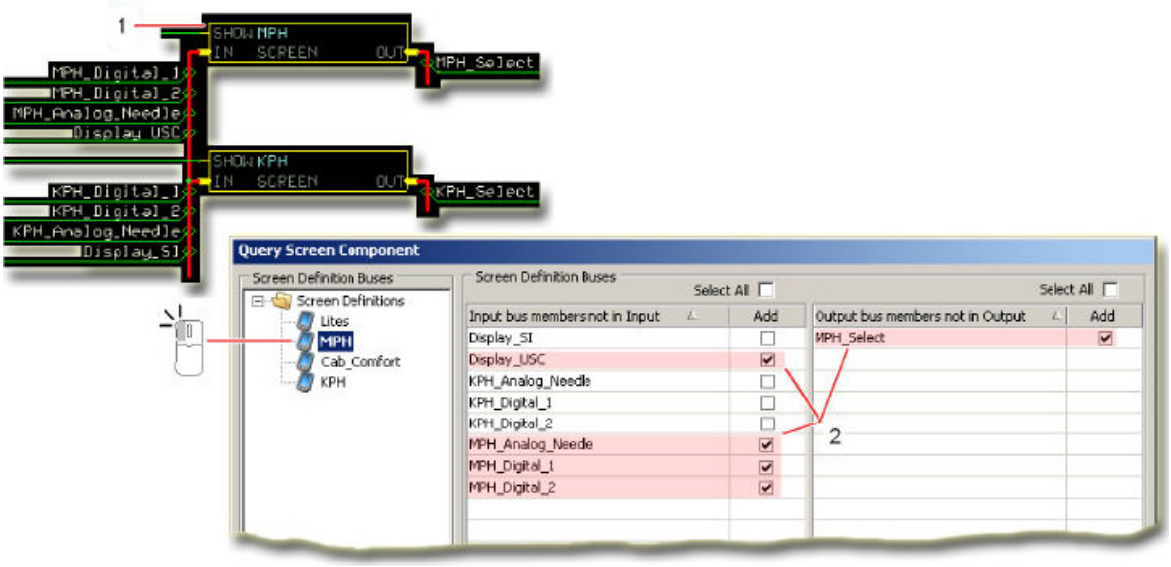
1. The **Show Screen** component that instances the **KPH Screen Definition** before you connect the:
   - **Overspeed_Warning** signal to this component's **IN** bus to resolve the **Definition bus members not in output bus** connection issue.
   - **Overspeed_Alarm** signal to this component's **OUT** bus to resolve the **Definition bus members not in output bus** connection issue.
2. The **Query Screen** window before you add the:
   - **KPH_Digital_1** and **KPH_Digital_2** signals to the Screen Definition to resolve the Input bus members not in Input connection issues.
   - **KPH_Select** signal to the Screen Definition to resolve the **Output bus members not in Output** connection issue.
3. The **Show Screen** component that instances the **KPH Screen Definition** after you connect the:
   - **Overspeed_Warning** signal to this component's **IN** bus to resolve the **Definition bus members not in output bus** connection issue.
   - **Overspeed_Alarm** signal to this component's **OUT** bus to resolve the **Definition bus members not in output bus** connection issue.
4. The **Query Screen** window after you add the:
   - **KPH_Digital_1** and **KPH_Digital_2** signals to the Screen Definition to resolve the Input bus members not in Input connection issues.
   - **KPH_Select** signal to the Screen Definition to resolve the **Output bus members not in Output** connection issue.

Screen Editors

About the Query Screen Component Window and MPH Screen Definition

The following figure shows what the **Query Screen Component** window displays after you:

- Resolve all signal connection issues for the **KPH Screen Definition**.
- Add a second **Show Screen** component to instantiate a similar **MPH Screen Definition**.

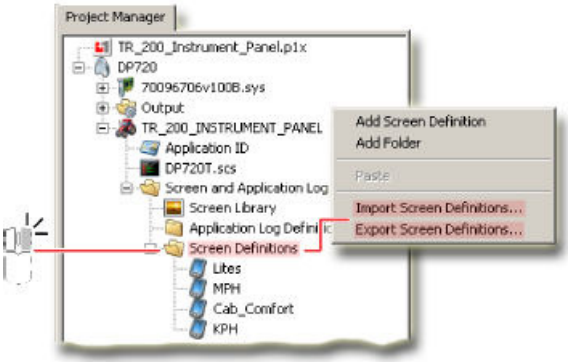


1. **Show Screen** component to instantiate the **MPH Screen Definition**
2. You must add the highlighted **Display_USC** and **MPH** signals to the **MPH Screen Definition** if you want to use them in the **MPH Screen Definition**.

   You do not need to add the **Display SI** and **KPH** signals to the **MPH Screen Definition** unless you also want to use them in the **MPH Screen Definition**.

Screen Editors

About Exporting and Importing Screen Definitions



Right-click in the Project Manager tab to open a pop-up menu with Export Screen Definition and Import Screen Definition commands.

- The **Import Screen Definition** command opens an **Open** window. Use this window to select an esd file for import.
- The **Export Screen Definition** command opens a **Save As** window. Use this window to export a screen definition file. Your screen definition exports to an exported screen definition file (*.esd). After selecting a esd file:
  - Use the**Import Screen Definition** — **Import Duplicate Library Items** window to manage the importation of a Screen Library with a name that duplicates that of an existing Screen Library already in the project. (The Import Screen Definitions—Import Duplicate Library Items window opens first if you have a duplicate Screen Library. Otherwise, the Import Screen Definitions—Screen Library Placement window opens first.)
  - Use the **Import Screen Definition** — **Screen Library Placement** window to manage the placement of imported Screen Library items in the Screen Editor window's Screen Library tab.
  - Use the**Import Screen Definition** — **Import Duplicate Library Items** window to manage the importation of any duplicate Screen Library items.

Screen Editors

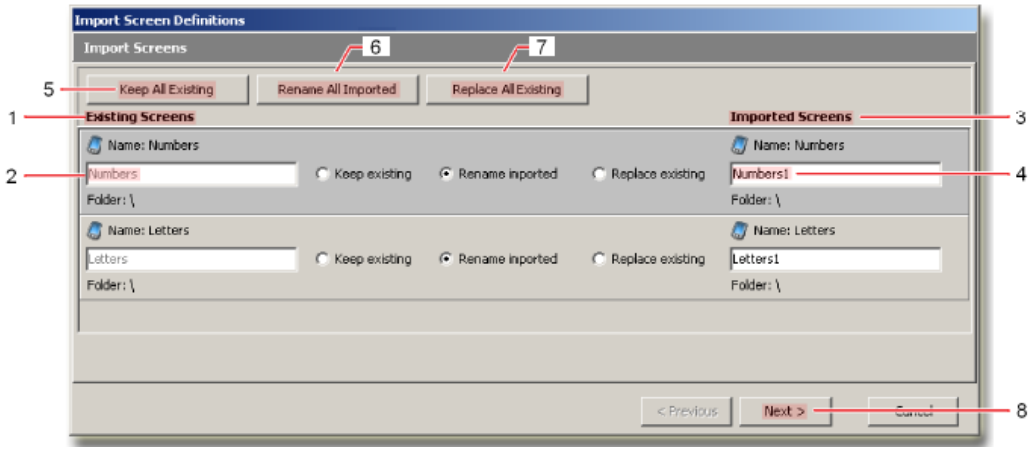
Import Screen Definitions—Import Screens Window

The **Import Screen Definitions—Import Screens** window opens first when you import a Screen Definition with a name that duplicates the name of a Screen Definition already in the project.

- Import the new Screen Definition under a new name.
- Or overwrite the existing Screen Definition with the imported Screen Definition.

Use the **Import Screen Definitions—Import Screens** window to either:

If you do not have a duplicate Screen Library, the **Import Screen Definitions—Screen Library Placement** window opens first, instead of the Import Screen Definitions— Import Screens window.

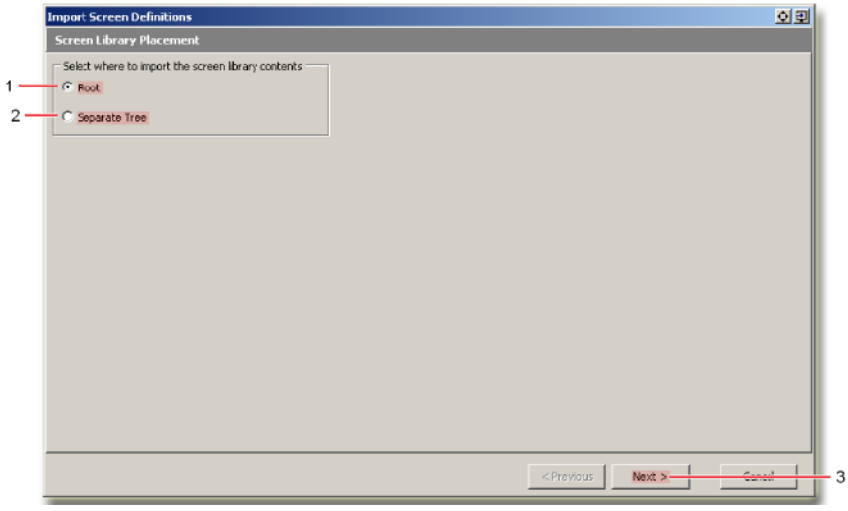


1. **Existing Screens**—lists each Screen Library in your project that the imported items will duplicate.
2. Existing **Screen Library** name—the name of the existing Screen Library that the imported Screen Library will duplicate.
3. **Imported Screens**—lists each duplicate Screen Library to be imported.
4. Imported **Screen Library** name—the name of the Screen Library after you import it.
5. **Keep All Existing** button—do not import any Screen Library. Use the individual **Keep Existing, Rename Imported**, and **Replace Existing** buttons to act on each duplicate Screen Library.
6. **Rename All Imported** button—import and rename every duplicated Screen Library. Use the Imported Items fields to rename a duplicated Screen Library. Use the individual **Keep Existing, Rename Imported**, and **Replace Existing** buttons to act on each duplicate Screen Library.
7. **Replace All Existing** button—overwrite each duplicate Screen Library. Use the individual **Keep Existing, Rename Imported**, and **Replace Existing** buttons to act each duplicate Screen Library.
8. **Next** button—click to open the **Import Screen Definitions—Screen Library Placement** window.

Screen Editors

Import Screen Definitions—Screen Library Placement Window

Use the **Import Screen Definitions—Screen Library Placement** window to manage the placement of imported **Screen Library** items in the **Screen Editor** window's Screen Library tab.

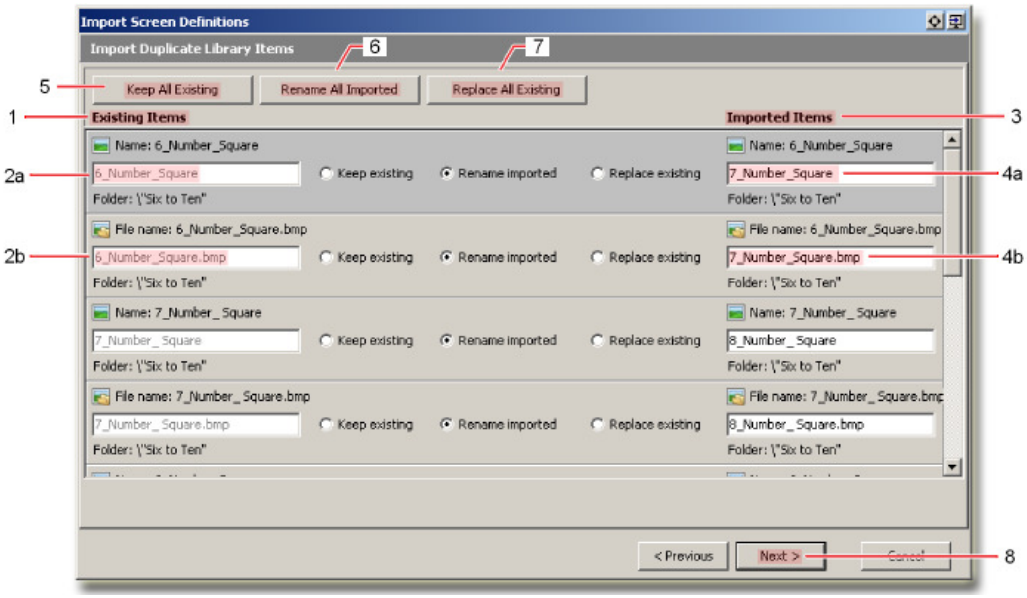


1. **Root** button—click to place items from the imported Screen Definition into the root directory of the Screen Library tab. This selection merges the imported items into the Screen Library tab's existing tree.
2. **Separate Tree** button—click to place items from the imported Screen Definition into their own subtree in the Screen Library tab.
3. **Next** button—click to either:
   - Automatically open the **Import Screen Definitions—Import Duplicate Library Items** window if you have duplicate Screen Library items.
   - Or, if you do not have duplicate Screen Library items, open the **Import Screen Definitions—Import** window to complete the import process.

Screen Editors

Import Screen Definitions—Import Duplicate Library Items Window

The **Import Screen Definitions—Import Duplicate Library Items** window opens if library items in your imported **Screen Definition** duplicate existing library items in your project.



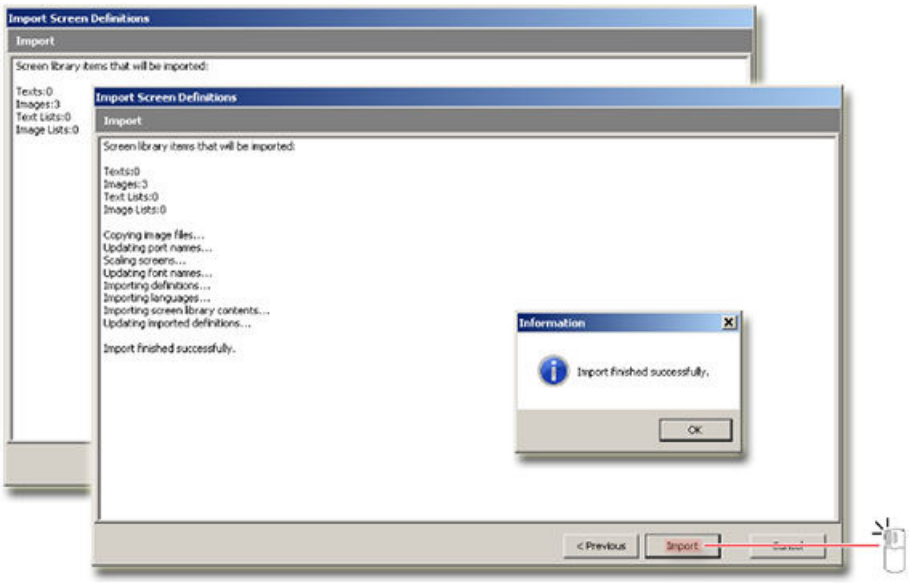
1. **Existing Items**—lists the library items in your project that the imported items will duplicate.
2. **a.** Existing **Screen Library** tab name—the name of the library item to be duplicated, as listed in the Screen Library tab.
   b. Existing file name—the file name of the library item to be duplicated, as listed in the project folder.
3. **Imported Items**—lists the duplicate library items to be imported.
4. **a.** Imported **Screen Library** tab name—the name of the duplicate item as it will be listed in the Screen Library tab after it imports.
   b. Imported project folder file name—the file name of the duplicate library item as it will be listed in the project folder after importing.
5. **Keep All Existing** button—import no library items. Use the individual **Keep Existing, Rename Imported**, and **Replace Existing** buttons to act on individual duplicate library items.
6. Rename All Imported button—import and rename all duplicated library items. Use the Imported Items fields to rename the duplicated library items. Use the individual Keep Existing, Rename Imported, and Replace Existing buttons to act on individual duplicate library items.
7. **Replace All Existing** button—overwrite all duplicated library items. Use the individual Keep Existing, Rename Imported, and Replace Existing buttons to act on individual duplicate library items.
8. **Next** button—click to open the **Import Screen Definitions—Import** window to complete the import process.

Screen Editors

Import Screen Definitions—Import Window

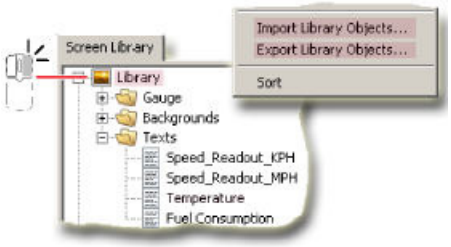
Use the **Import Screen Definitions—Import** window to:

- Review what you will import.
- Complete the importation process.
- Review what you have imported.

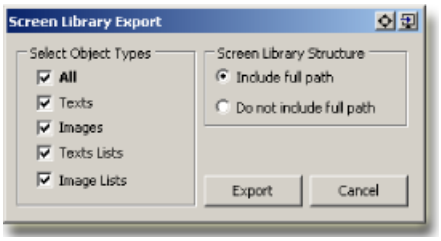


Screen Editors

About Exporting and Importing Library Objects



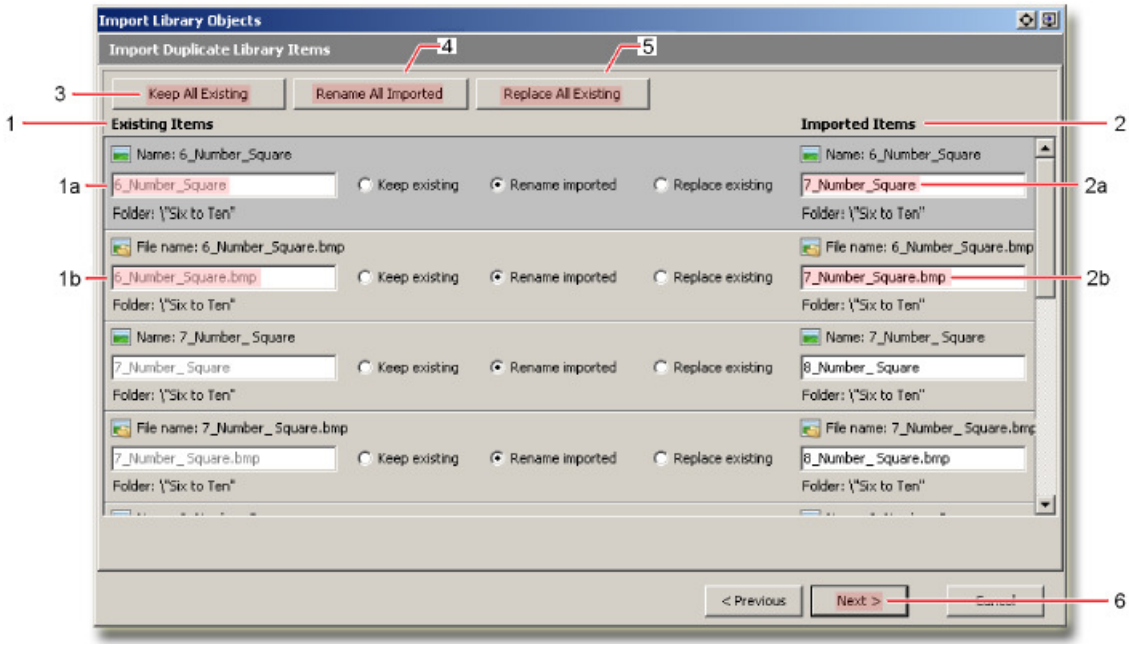
Right-click in the **Screen Library** tab of a **Screen Definition** to open a pop-up menu with **Export Library Objects** and **Import Library Objects** commands.



- The **Export Library Object** command opens a **Screen Library Export** window. Use this window to select what object types export and what Screen Library tab tree structure exports with the selected objects. Your library objects export to an exported library definitions file (*.eld).
- The **Import Library Object** command opens an Open window. Use this window to browse to and select an eld file.
  - The **Import Library Objects—Import Duplicate Library Item**s window opens if your eld file contains duplicate library object items. Use this window to manage the importation of duplicate library object items.

Screen Editors

Import Library Objects—Import Duplicate Library Items

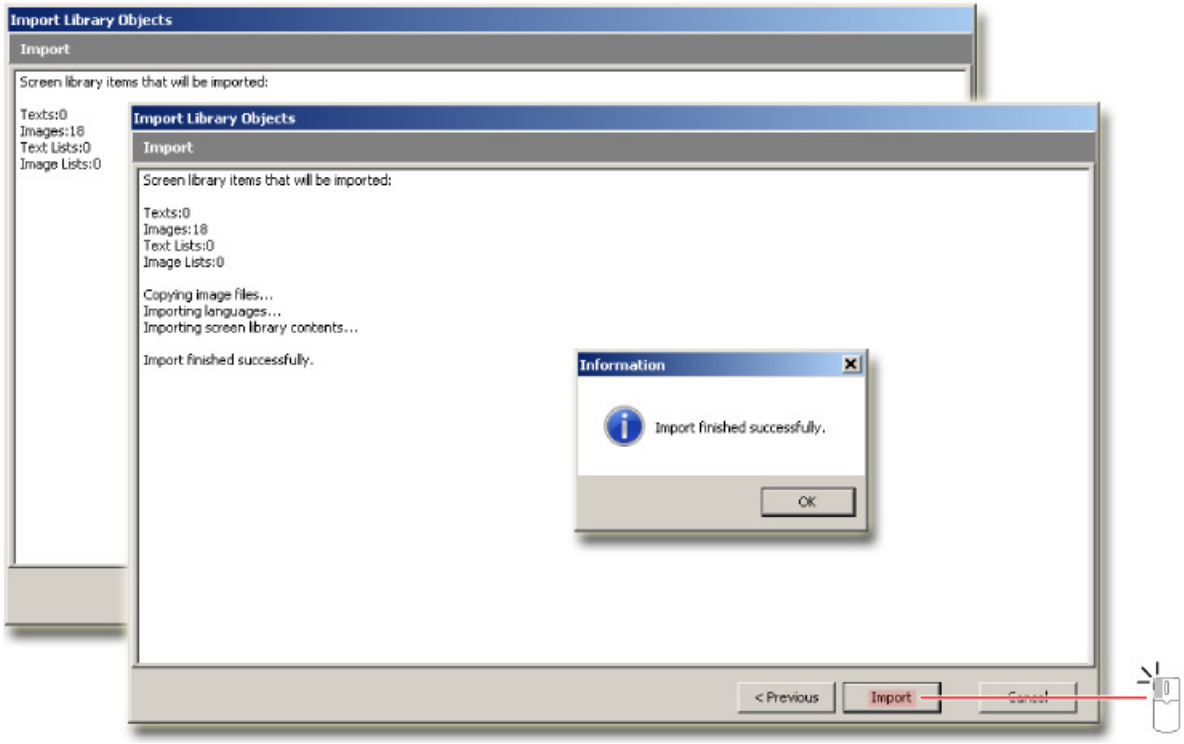


1. **Existing Items**—lists the library items in your project that the imported library items will duplicate.
   a. Existing **Screen Library** tab name—the library object name of the library item to be duplicated, as listed in the Screen Library tab.
   b. Existing file name—the file name of the library item to be duplicated, as listed in the project folder.
2. **Imported Items**—lists the duplicate library items to be imported.
   a. Imported **Screen Library** tab name—the library object name of the duplicate library item as it will be listed in the Screen Library tab after importing.
   b. Imported project folder file name—the file name of the duplicate library item as it will be listed in the project folder after importing.
3. **Keep All Existing** button—import no library items. Use the individual **Keep Existing, Rename Imported**, and **Replace Existing** buttons to act on individual duplicate library items.
4. **Rename All Imported** button—import and rename all duplicated library items. Use the Imported Items fields to rename the duplicated library items. Use the individual **Keep Existing, Rename Imported**, and **Replace Existing** buttons to act on individual duplicate library items.
5. **Replace All Existing** button—overwrite all duplicated library items. Use the individual **Keep Existing, Rename Imported**, and **Replace Existing** buttons to act on individual duplicate library items.
6. **Next** button—click to open the **Import Library Objects—Import** window to execute the import process.

Screen Editors

Import Screen Definitions—Import Window

Use the **Import Screen Definitions—Import** window to review what will be imported and to execute the importation process.



Touch Display Functionality

Touch display functionality is hardware dependent. The features described in this topic are available only if the HWD of the currently opened project supports touch display functionality.

All objects in the **Design Area** have touch related properties. These properties can be found in the **Touch** branch in the **Inspector** or the **Touch** tab of the object's **Common Properties** dialog. There is also a screen object type called **Touch Area** that can be used for detecting touches within a rectangular area. **Touch areas** will not affect the appearance of the display.

To add a **Touch Area** to a screen definition, drag it from the **Selector** and drop in the **Design Area**.

Touch events are detected within the boundaries of each component. Detection is processed in front-to-back display order; for example, if two objects overlap the object on top will detect the touch event first. Touch events are output with a delay of one program loop.

When a hardware unit has multiple touch streams and both indicate a touch inside a certain object, only one of them will be detected by the object.

If the hardware unit supports multiple touch streams and gestures, the **Touch node** will be called **Touch** (**MultiTouch**) in the inspector and contain additional outputs. Refer to the API specification of the HWD for a description of the outputs **GestureType**, **DeltaX**, **DeltaY**, **RotationAngle**, **ScaleFactor** and **Velocity**.

LocalDeltaX and **LocalDeltaY** are described in *Local Touch Coordinates* on page 526.

Touch Propagation

Screen definitions can be set up to allow touches to propagate to the screen definition beneath it, if any. Use property **Propagate Touches** to control this behavior. Default value is **False** (touches will not propagate).

Screen Editors

A widget instance will behave according to the value of **Propagate Touches** in its definition.

Within a screen definition, each screen object can be set up to block touches *within that screen definition*. Use property **IsSink** to control this behavior. Default value is **False** (touches will propagate).

Warning

Screen definitions saved by a PLUS+1 GUIDE version prior to 2021.2 have no **Propagate Touches** property; touches would always propagate beneath a screen definition. To keep backwards compatibility, the default value of **Propagate Touches** in screen definitions saved in older formats is **True**. (This also applies to exported screen definitions and library widgets.)

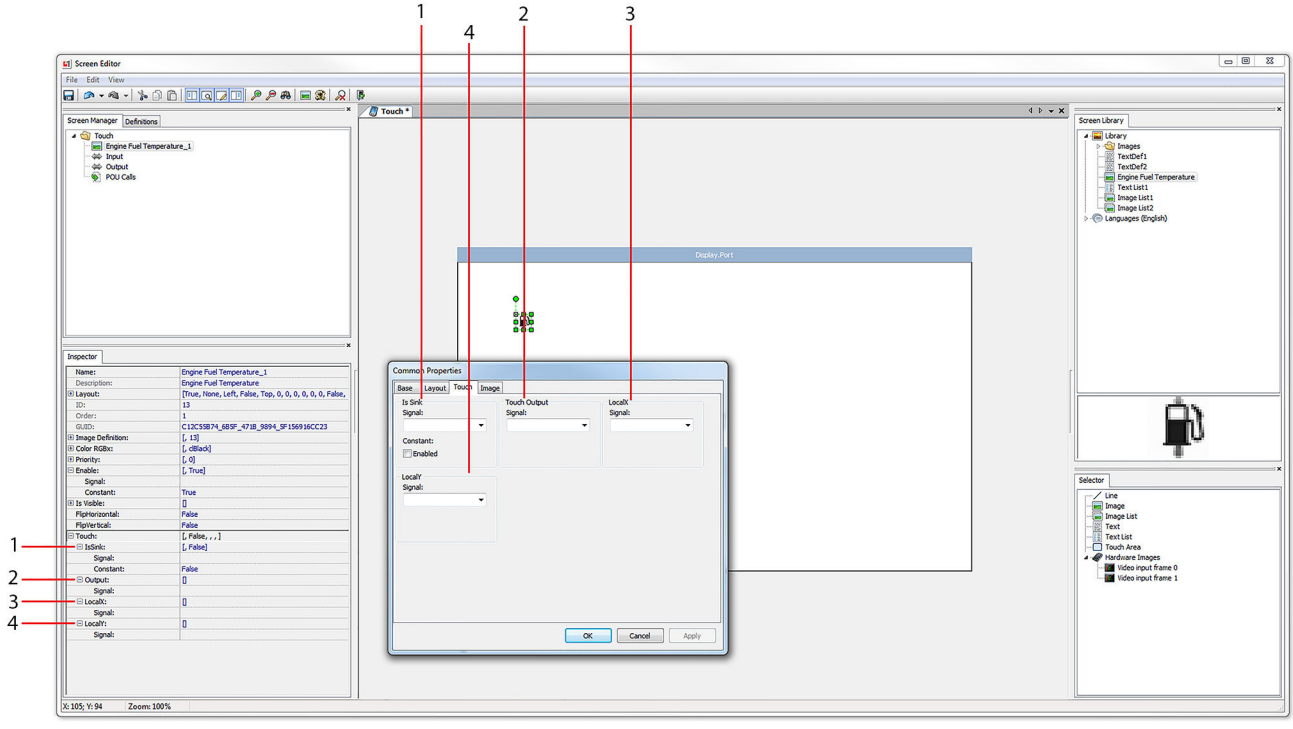
When opening project containing a screen repository saved by a PLUS+1 GUIDE version prior to 2021.2, a dialog will be shown to set the value of **Propagate Touches** for all screen definitions in the project.

- Choose ***Propagate*** to set **Propagate Touches** to **True** and keep the behavior of the old version of PLUS+1 GUIDE.
- Choose ***Don't Propagate*** to set **Propagate Touches** to **False** and stop touches from propagating beneath a screen definition.

Screen Object Touch Properties

This section describes the screen object touch properties and touch states.

Screen object touch properties



Screen Editors

Touch properties

Item	Name	Description
1	**IsSink**	Controls whether this screen component will block touches within its screen definition. • Set to **True**: Any touch detected on this component will not propagate to underlying components. • Set to **False**: Any touch detected on this component will propagate to underlying components. **IsSink** will only block touches that are detected. If two streams indicate touches on an object having **IsSink** set to true, only the touch that is detected will be blocked. **IsSink** does not apply to gestures. When a gesture is detected, any objects beneath the detecting object in the same screen definition will not detect the same gesture. The streams involved in the gesture will be available for objects beneath the detecting object.
2	**Touch Output**	• Any touch activity detected within the boundaries of the component since last program loop: **Touch Output** will be set to the state corresponding to the touch event. A list of states can be found in the API specification. • No touch activity was detected within the boundaries of this component since last program loop: **Touch Output** will be set to the value corresponding to the idle state.
3	**LocalX**	The X coordinate of the touch activity, in the coordinate system of the screen object. See *Local Touch Coordinates* on page 526.
4	**LocalY**	The Y coordinate of the touch activity, in the coordinate system of the screen object. See *Local Touch Coordinates* on page 526.

Touch States

Touch States

N	State	Active for multiple loops	Active for only one loop	Max time	Description
0	Idle State (No Touch)	x			Idle State will be active after an activity and no Touch Active (touch on the screen).
1	Touch Active	x			
2	Short Touch Release		x	In milliseconds	A Short Touch will be activated on same position, when the touch event is released before the max time.
3	Double Click		x	The duration between touch events in milliseconds	
4	Long Press Release		x		The Long Press Release activates on the max time of Short Touch Release at same position.
5	Swipe Left		x		
6	Swipe Right		x		
7	Swipe Up		x		
8	Swipe Down		x		

Local Touch Coordinates

Each touchable screen object has its own coordinate system. The origin of this coordinate system is in the upper left corner of the object; the X axis is directed to the right along the top edge of the object and the Y axis is directed down along the left edge of the object. On hardware units having one touch stream, local touch coordinates will be output even if the touch output of the object indicates an idle state. Local touch coordinates are undefined on hardware units having multiple touch streams when the touch output indicates an idle state.

Hardware units supporting multiple touch streams have two additional output signals **LocalDeltaX** and **LocalDeltaY**. They will indicate touch movement along the local X and Y axes of the screen object.

- The local coordinate system will have the same rotation as the screen object.
- Insertion point offsets do not affect the local coordinate system. Its origin will be in the top left corner even if the screen object has an insertion point offset. This is relevant for image objects, hardware image, and generic viewport objects.
- The local coordinate system of a line will be in the top left corner when the **Begin point** is located to the left of the **End point** and the line is horizontal.

Refer to the documentation of the hardware unit to get the number of touch streams.

Generic Viewport

Use Generic Viewport to display graphical content such as video files. The Generic Viewport object will specify spatial properties of the graphical content such as size, position and rotation. All properties may not be relevant for all types of graphical content. Additional properties, such as media file name, are specified by other means, e.g. API variables. Generic Viewport is tied to the graphical content via its UIDOutput property.

Generic Viewport requires HWD support. Refer to the HWD documentation for details on the available kinds of graphical content and how to set them up.

Application Data Logging

Use application data logging with your applications to capture information about how your hardware components are operating.

Overview of Application Data Logging

Application data logging is hardware dependent. Not all PLUS+1® controllers support application data logging. Application data logging writes data to the memory of a Danfoss PLUS+1® controller. The PLUS +1® Service Tool program accesses this data. The PLUS+1® Service Tool program writes data first to an encrypted P1A (PLUS+1® application data log) file and then, with proper access rights, to a CSV (comma-separated values) file.

There are two application log functionalities in PLUS+1® GUIDE:

- Classic Application Log using the Classic Screen Editor
- Application Log 2 using the Vector Based Screen Editor

Each HWD that supports application log supports exactly one of these functionalities.

The **Access App Log Enable** component must be used to enable access by the free version of the PLUS +1® Service Tool program to an application data log.

Warning

Without Tool Key protection, there is an increased risk that unauthorized personnel could use the PLUS +1® Service Tool program to view your application's data log and to change your application's operating parameters. Changes in your application's operating parameters could cause unexpected machine movements that result in personal injury and equipment damage. Always use the PLUS+1® GUIDE program's Tool Key feature to restrict access to your application's operating parameters. Tool Key protection reduces the risk that unauthorized personnel could view and change your application's operating parameters.

Refer to *Use the Tool Key to Restrict Service Tool Access to Application Values*.

Classic Application Log

Classic Application Log using the Classic Screen Editor

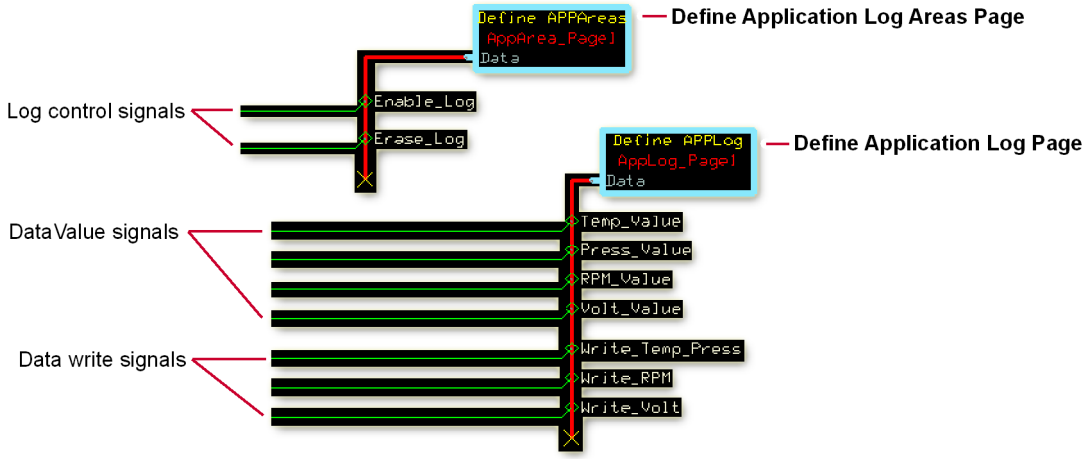
Basic Elements of Application Data Logging

Define Application Log Areas Page and the Define Application Log Page implement basic application data logging.

- The **Define Application Log Areas Page**:
  - Receives Log control signals that both enable application data logging and erase the contents of the application data log.
  - Reserves controller memory for the data application log and sets the type of log as either circular or linear.
- The **Define Application Log Page**:
  - Receives DataValue signals with values that write to the application data log.
  - Receives Data write signals that determine when values write to the application data log.
  - Tags data by category and access level.

An application can have multiple application data logs. This figure does not show the logic for the Log control, DataValue, or Data write signals.

Application Data Logging

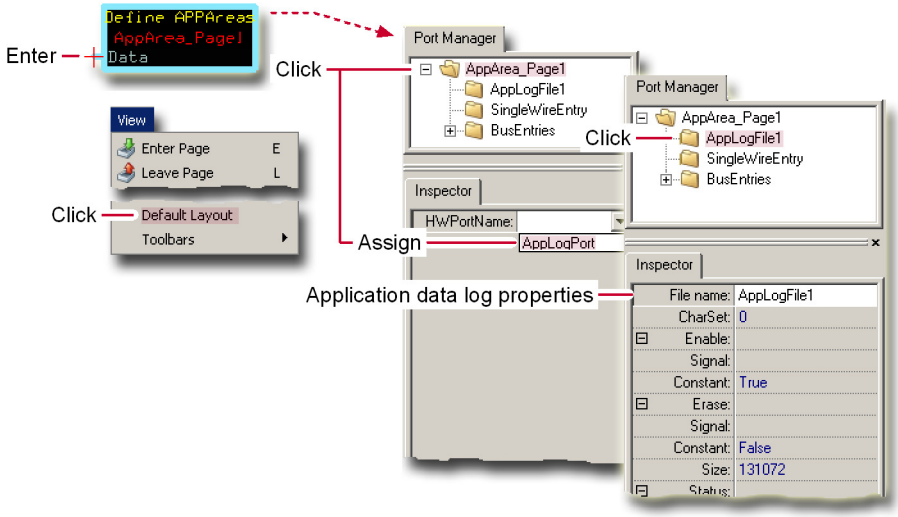


Basic Elements of Application Data Logging

Item	Description
Define Application Log Areas Page	Reserves controller memory for the application data log. Defines: • the application data log type as either circular or linear. • when the controller can log data to the application data log. • when the controller erases the contents of the application data log.
Define Application Log Page	Defines what and when data writes to the application data log.

Application Data Logging

Define Application Log Areas Page



This figure shows elements that display when you enter the **Define Application Log Areas Page** and select the **Default Layout** view.

The **Define Application Log Areas Page**:

- reserves controller memory for the application data log.
- defines when the controller can log data to application data log.
- defines when the controller erases the contents of the application data log.
- defines the application data log type as either circular or linear.

The Screen Library tab also displays but is not used in the **Define Application Log Areas Page**.

The **Define Application Log Areas Page** does not support sub-buses. Always use a main bus to bring signals to the **Define Application Log Areas Page**.

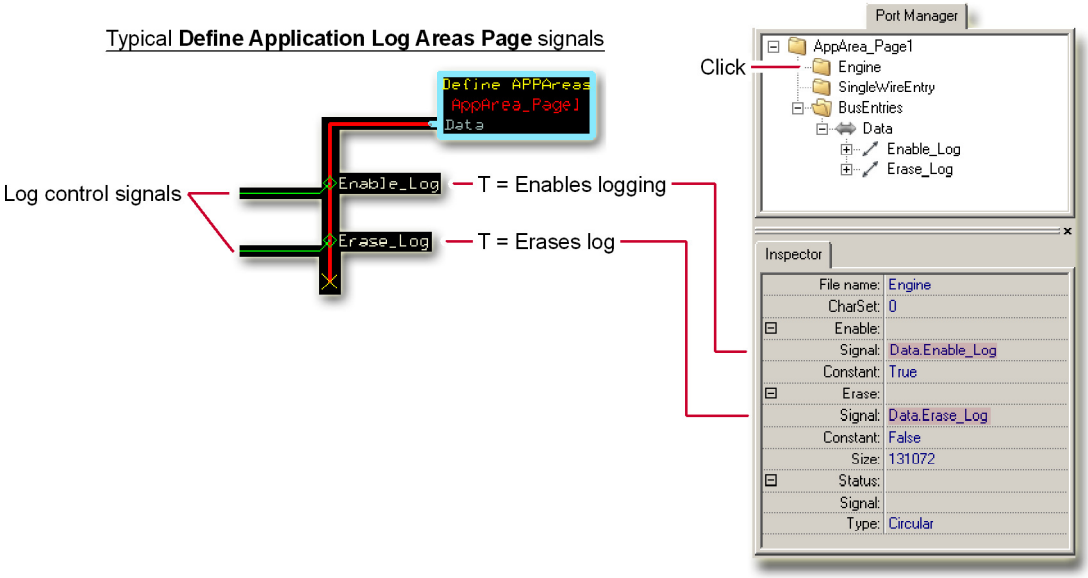
Define Application Log Areas Page

Item	Description
Port Manager tab	Use this tab to manage the properties of the application data log. • **AppArea_Page1** folder—click to display the **HWPortName** used by the application data log. • **AppLogFile1** folder—the default file name for the application data log. The PLUS+1 Service Tool accesses the application data log through this name. Use the **Inspector** tab to change this name. • **SingleWireEntry** folder—not used here. • **BusEntries** folder—click for a tree view of signals routed to the **Data** input of the **Define Application Log Areas Page**.
Inspector tab	Use this tab to view and manage the properties of the application data log.
Buttons	The size of the PLUS+1 GUIDE window determines the row in which these buttons appear.
	Exit Screen Editor—click to exit the **Define Application Log Areas Page**.
	Help Contents—opens the PLUS+1 GUIDE Help.

Application Data Logging

Define Application Log Areas Page/Inspector Tab

Use the **Inspector** tab to manage the properties of the application data log.



Inspector Tab elements description

Item	Description
File name	The PLUS+1 Service Tool accesses the application data log through this name. Enter a user-friendly, meaningful name here.
CharSet	Defines the character set of data written to the application data log. • Set this value to **0** for data that uses Western, Roman alphabets. • Enter the required character set number for data that uses a non-Roman alphabet.
Enable	When True, data can write to the application data log. This property must be True before any data can write to the application data log. A **Signal** input overrides a **Constant** value. • True = Data can write to the application data log • False = No data can write to the application data log
Erase	When True, erases the contents of the application data log. A **Signal** input overrides a **Constant** value. True = Erase the contents of the application data log
Status	Selects the U16 signal that outputs application data logging status. • Bit 1 (0x0001) = writing data to the application data log. • Bit 2 (0x0002) = writing the contents of the application data log to an application data log file. Pending data writes to the application data log when the read process ends. • Bit 3 (0x0004) = erasing the contents of the application data log. Pending data writes to the application data log when the erase process ends. • Bit 4 (0x0008) = the application data log is full. Applies only to application data logs defined as linear.
Size	Sets the memory size of the application data log. Refer to the controller's API specification for more about memory allocation for an application data log.
Type	Defines the application data log as either **Circular** or **Linear**.

Application Data Logging

About the Enable and Pending Signals

The following table shows the relationship between the **Enable** and **Pending** signals in writing data to an application log.

You access the:

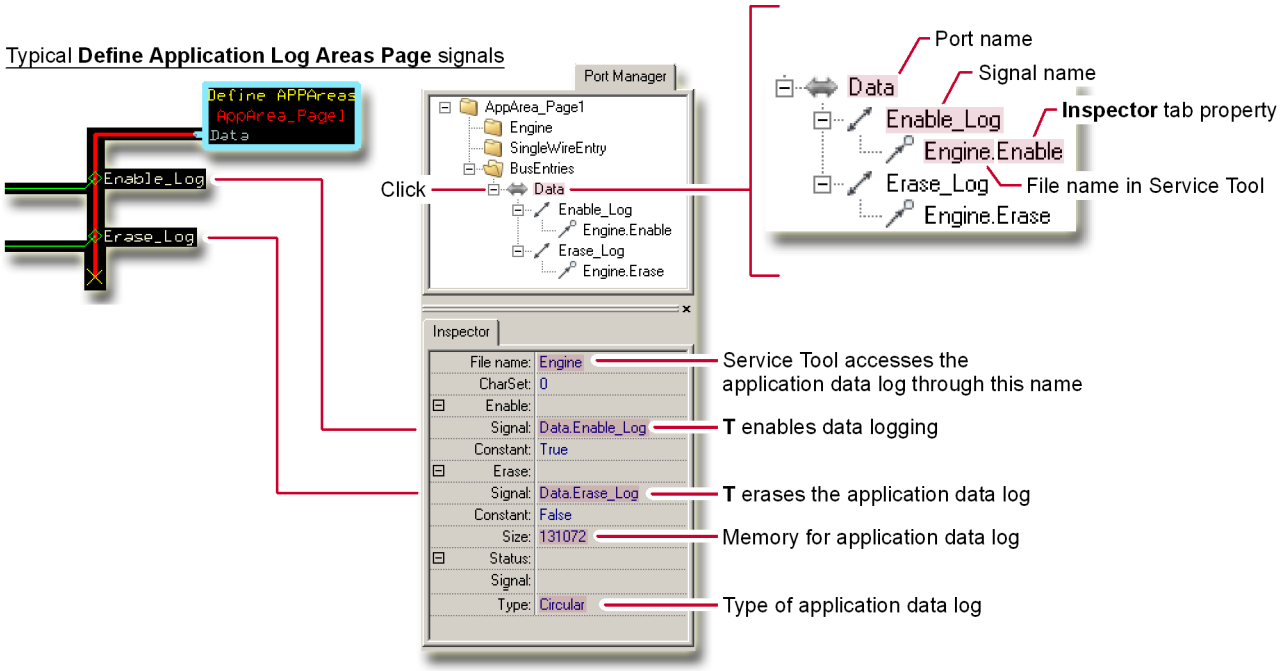
- **Enable** signal in the **Inspector** tab of the **Define Application Log Areas** page.
- **Pending** signal in the **Inspector** tab of the **Define Application Log** page.

Enable and Pending Signals

Write Signal State	Pending Signal State	Action
T	F	Data writes to the application log.
F/T	T	No new data writes to the application log.

Application Data Logging

Define Application Log Areas Page/About the Port Manager and Inspector Tabs



This figure shows:

- A **Define Application Log Areas Page** with Boolean **Enable_Log** and **Erase_Log** signals routed to its **Data** input.
- The **Port Manager** and **Inspector** tabs of Define **Application Log Areas Page**, with the:
  - **Enable_Log** signal selected to control the **Enable** property.

    The **Enable_Log** signal must be True before any data can write to the application data log.
  - **Erase_Log** signal selected to control the **Erase** property.

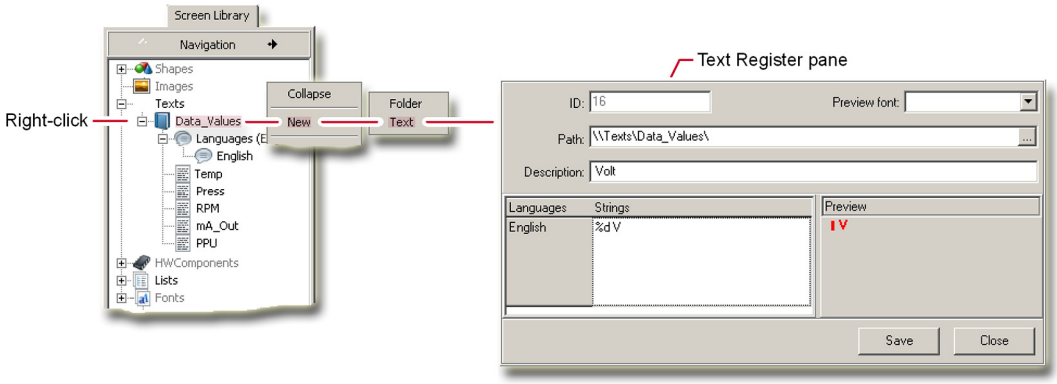
    The **Erase_Log** signal, when True, erases the contents of the application data log.
- How the **BusEntries** (signals) to the page are mapped to the **Inspector** tab's properties.

Application Data Logging

Define Application Log Page/Text Register tab

Use the **Text Register** tab to create **Texts** for the **Screen Library** tab.

Creating **Texts** defines the strings that can be written to the application data log.

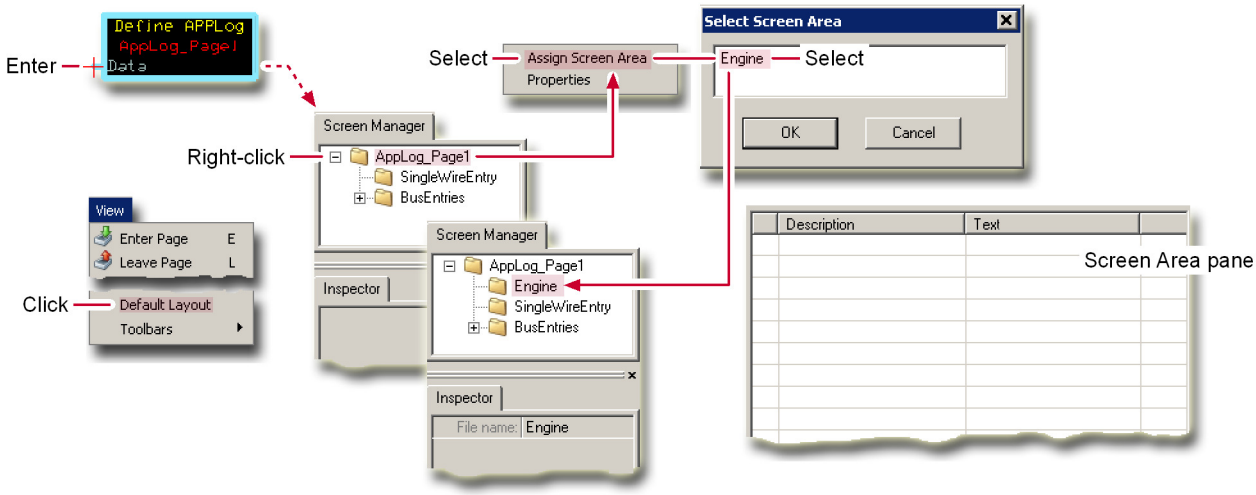


Define Application Log Page/Text Register tab

Item	Description
ID	Used for identification purposes. You cannot change this value.
Preview font	Not used.
Path	The location the item in the **Screen Library** tab's **Texts** tree.
Description	The descriptive name for the **Texts** item. The name entered here displays in the **Screen Library** tab's **Texts** tree and, if you drag the item to the Screen Area pane, in the **Screen Manager** tab's application data log folder.
Languages	Identifies the language of the **Texts** item.
Strings	The string entered writes to the application data log. Use the **%** print character to display data values in the string.
Preview tab	Previews **Strings** entries as they appear in the application data log.
Save	Closes the Text Register tab and adds the **Texts** to the **Screen Library** tab's **Texts** tree and to the **Screen Manager** tab's application data log folder.
Close	Closes the Text Register tab.

Application Data Logging

Define Application Log Page



This figure shows elements that display when you enter the **Define Application Log Page**, select the **Default Layout** view, and assign a Screen Area to the selected application data log.

The **Define Application Log Page** does not support sub-buses. Use the main bus to bring signals into this page.

The **File** menu's **Import Page** and **Import Block** commands do not work with this component. Importing a **Define Application Log Page** strips this page of its contents.

Define Application Log Page

Item	Description
Screen Manager tab	Displays a folder for the application data log (the **Engine** folder in this figure).
Screen Area tab	Lists **Texts** (values) that write to the application data log. An empty pane displays as soon as you assign a Screen Area to the selected application data log.
Inspector tab	Manages the properties of values that write to the application data log.
Buttons	The size of the PLUS+1 GUIDE window determines the row in which these buttons appear.
	Exit Screen Editor—click to exit the **Define Application Log Page**.
	Help Contents—opens the PLUS+1 GUIDE Help.

Application Data Logging

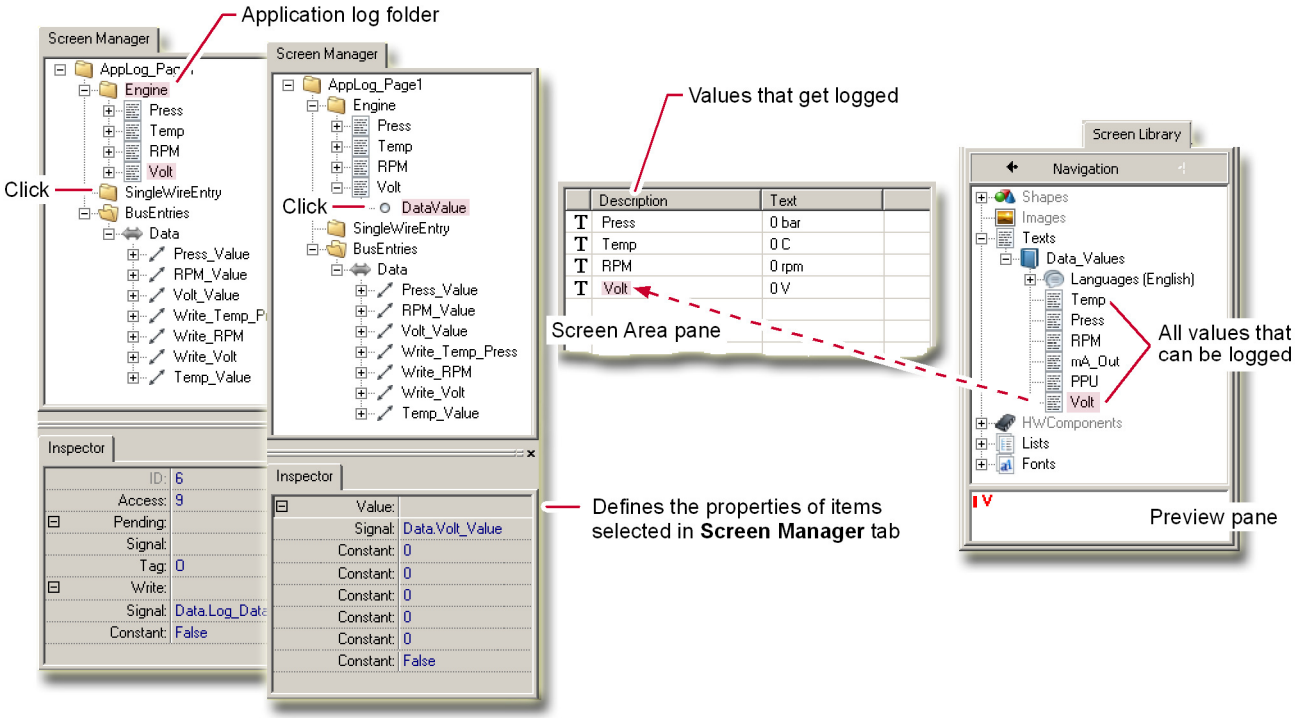
Define Application Log Page/Add Texts

The contents of the application data log folder (the **Engine** folder in the figure below) define:

- What data writes to the application data log.
- When the data writes to the application data log.

You define the contents of the application data log folder by:

- Dragging **Texts** for the values to be logged from the **Screen Library** tab into the Screen Area pane.
- Using the **Inspector** tab to define the properties of the **Texts**.



Define Application Log Page/Add Texts

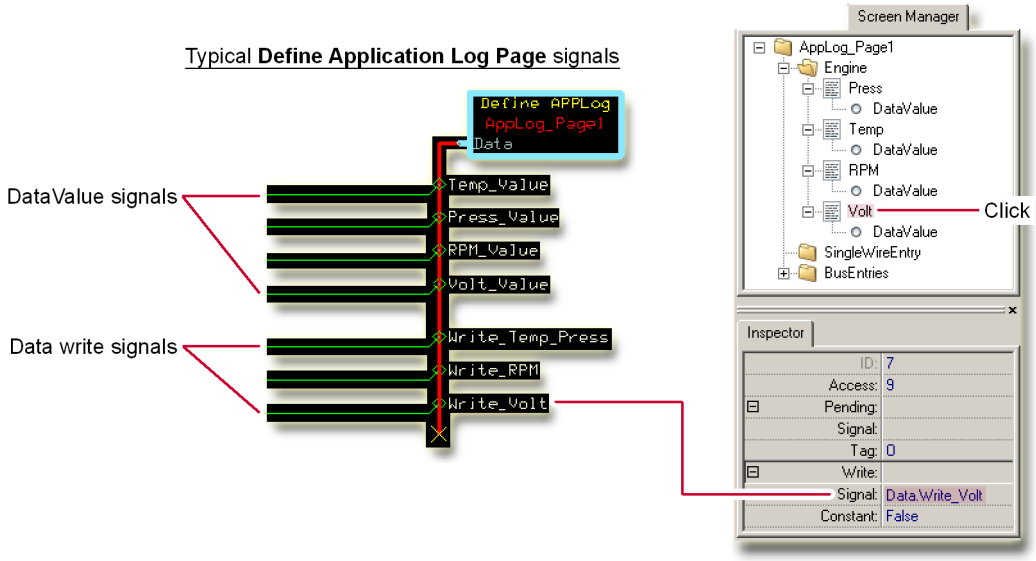
Item	Description
Screen Manager tab	• **AppLog_Page1** folder — click for a tree view of the properties of **APPLog_Page1**. Application log folder (the **Engine** folder in this figure)—click for a tree view of **Texts** added to the Screen Area. • **SingleWireEntry** folder — not used here. • **BusEntries** folder — click for a tree view of all signals routed to the **Data** input of the **Define Application Log Page**.
Inspector tab	Defines the properties of items selected in the **Screen Manager** tab.
Screen Area tab	Lists the **Texts** whose values write to the application data log. The order in which **Texts** appear in the **Description** column of this tab sets the order in which their values appear in the application data log.
Screen Library tab	Displays a tree view of **Texts** whose values can be written to the application data log. To select values to be written to the application data log, drag **Texts** from the **Screen Library** tab into the Screen Area tab.
Preview tab	Displays a preview of **Texts** values as they appear in the data log file.

Application Data Logging

Define Application Log Page/Inspector Tab—Data Write Properties

Data Write Properties define:

- when data writes to the application data log,
- the category and access level of the data.



Inspector Tab—Data Write Properties

Item	Description
ID	A read-only value used for internal identification purposes.
Access	Sets the access level of the data. This value works with **Access** components to limit users' access to the application data log with the PLUS+1 Service Tool.
Pending	Selects the Boolean output signal that indicates if data is ready to write. True = Data is ready to write to the application data log
Tag	Sets a category for the data. This value works with **Access** components to limit users' ability to access to categories of data in the application data log with the PLUS+1 Service Tool. • **E** = Error • **S** = Statistics • **O** = Other
Write	When True, writes the data to the application data log. A **Signal** input overrides a **Constant** value. True = Write the data to the application data log.

Application Data Logging

About the Enable and Pending Signals

The following table shows the relationship between the **Enable** and **Pending** signals in writing data to an application log.

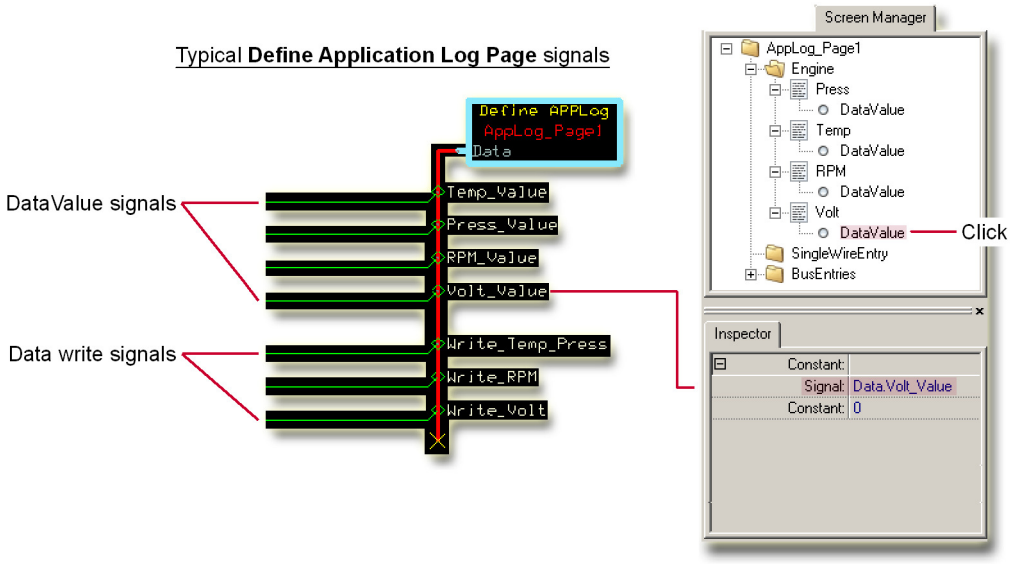
You access the:

- **Enable** signal in the **Inspector** tab of the **Define Application Log Areas** page.
- **Pending** signal in the **Inspector** tab of the **Define Application Log** page.

Enable and Pending Signals

Enable Signal State	Pending Signal State	Action
T	F	Data writes to the application log.
F/T	T	No new data writes to the application log.
T	T/F	Data writes to the application log.

Define Application Log Page/Inspector Tab—DataValue Properties



DataValue properties define the value written to the application data log.

Constant selects the signal whose value writes to the application data log.

Application Log 2

Application Log 2 using the Vector Based Screen Editor

Application Log Definitions

Application log data is written by instances of application log definitions. There can be multiple application log definitions and each application log definition can be written at several locations in the PLUS+1® GUIDE application.

Compared to the older application log functionality, Application Log 2 is fundamentally different. Conceptually, **Application Log Areas** page and **Application Log** page are merged into one entity called **Application Log Definition**. Application Log Definitions are defined in a repository that is accessed from the Project Manager. A component called Write Applog is used to invoke an instance of an Application Log Definition. Each Write Applog component that is used in a PLUS+1® GUIDE project will result in the creation of one instance of an Application Log Definition.

Application Data Logging

Warning

All text definitions in Application Log 2 and Vector Based Screen Editor are encoded in Unicode. PLUS+1® Service Tool versions prior to 9.0 reads application log data in ANSI.

Write Applog

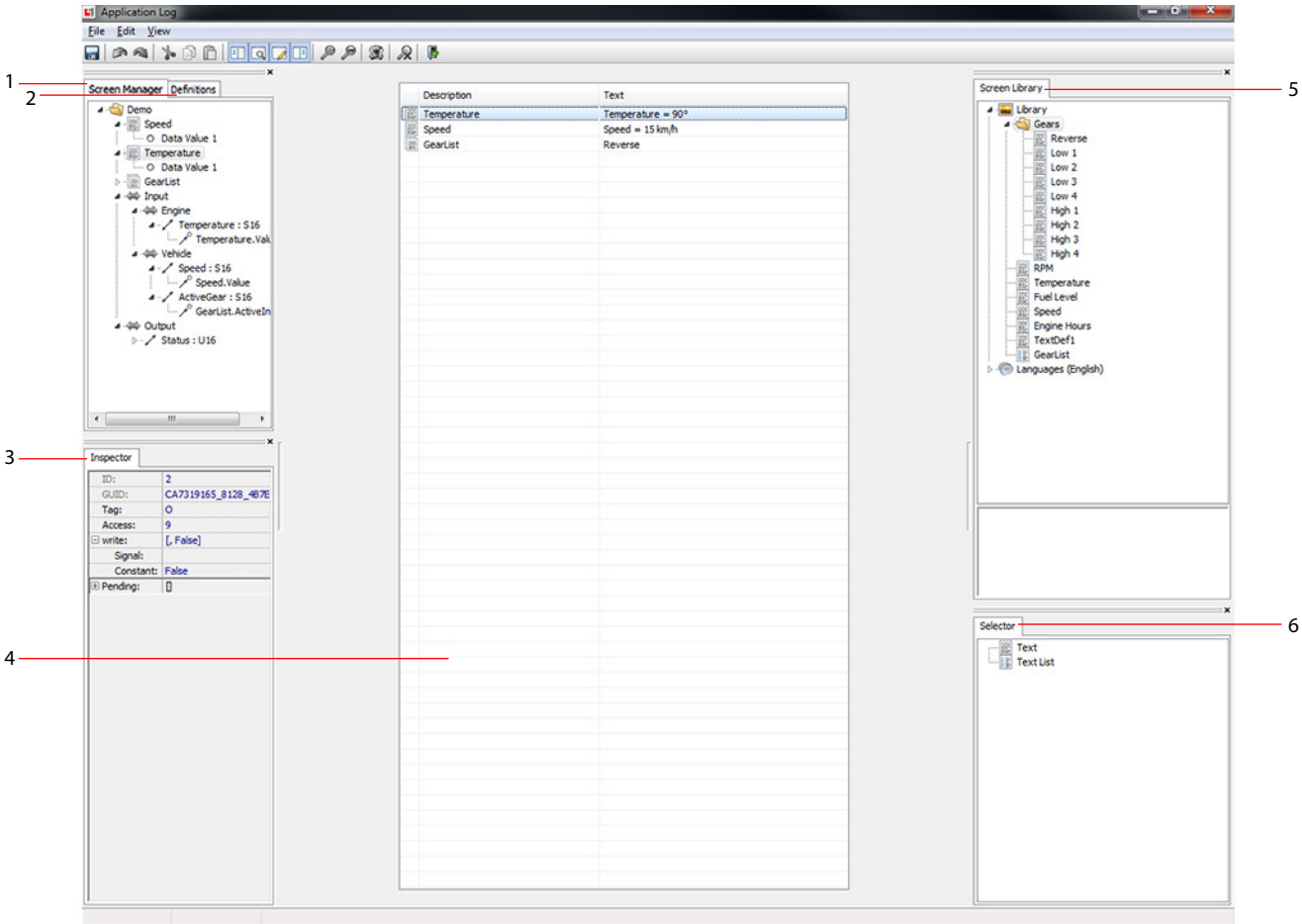
The Write Applog component is used for invoking an Application Log Definition. For information, see *Write Applog* on page 377.

Application Log 2 Editor

Application Log 2 Editor is based on Vector Based Screen Editor. The appearance of Application Log 2 Editor is therefore similar to the appearance of Vector Based Screen Editor with three notable exceptions:

- The **Design Area** is replaced with a **Screen Area Pane** where the texts to log are placed.
- The Screen Library only contains Texts and Text Lists. If the current HWD also supports Screen Definitions, then all Texts and Text Lists in the Screen Library are shared between Application Log Definitions and Screen Definitions.
- The properties of Texts and Text Lists are different in the Application Log 2 Editor and the properties of an Application Log Definition do not match the properties of a Screen Definition.

Application Log 2 Editor Window



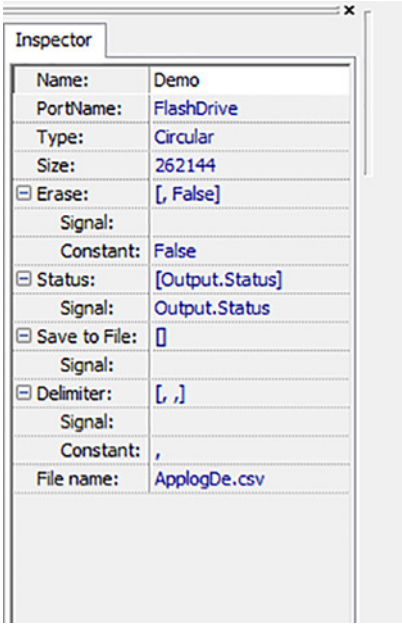
Application Data Logging

Application Log 2 Editor Elements

Item	Name	Description
1	Screen Manager	The Screen Manager tab displays the content of the currently edited Application Log Definition. By selecting a node, its properties can be edited in the inspector. The root node contains the properties of the Application Log itself; its sub-nodes consist of all elements present in the Screen Area Pane. Finally, there are two sub-nodes called Input and Output. These two nodes contain the interface of the Application Log Definition.
2	Definitions	Use this tab to view and modify the contents of the Application Log Definitions branch or to switch Application Log Definition being edited.
3	Inspector	Use this tab to modify the properties of the currently selected object in the Screen Manager.
4	Screen Area Pane	The Screen Area Pane contains all Texts and Text Lists of the Application Log Definition. Objects are added to the Screen Area Pane by dragging them from the Screen Library.
5	Screen Library	The Screen Library contains all Texts and Text Lists of the currently opened PLUS+1® GUIDE project. Use this tab to add, edit or remove Texts and Text Lists. Note, other kinds of objects such as images are not displayed in the Application Log 2 Editor even though they exist in the Screen Library.
6	Selector	By dragging an object from the Selector to the Screen Area Pane, a new object is added to the Application Log Definition and to the Screen Library.

Application Log Definition properties

The Application Log Definition manages the memory of an application log and contains all Texts and Text Lists that can be written to it.



Application Log Definition Properties

Item	Description
Name	This is the name of the Application Log Definition.
PortName	Select which memory area to use in the hardware unit.
Type	Defines the application data log as either Circular or Linear. If Circular is selected, the oldest data will be overwritten when the application log memory allocated to this Application Log Definition instance is full. If Linear is selected and the application log memory allocated to this Application Log Definition instance is full, no more data will be written until this application log instance is erased.

Application Data Logging

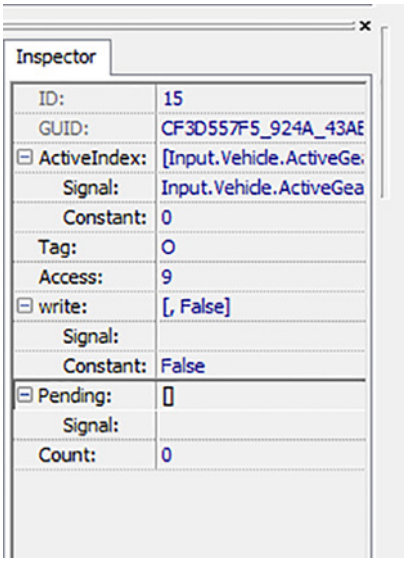
Application Log Definition Properties (continued)

Item	Description
Size	Size of the Application Log Definition. The size will be adjusted automatically to a multiple of the memory sector size in the hardware unit.
Erase	Erase the contents of the Application Log Definition if all conditions below are met: • No write is ongoing • No read is ongoing • No erase is ongoing • No save to file is in progress
Status	Current status of the Application Log Definition instance. It consists of a bit field with the following bits: Bit 1 (0x0001): Write is ongoing. Bit 2 (0x0002): Read is ongoing. Pending data writes to the application data log when the read process ends. Bit 3 (0x0004): Erase is ongoing. Pending data writes to the application data log when the erase process ends. Bit 4 (0x0008): End of file has been reached. Applies only to application log definitions that have the Type property set to Linear. Bit 5 (0x0010): Save to file has failed.[1] Bit 6 (0x0020): Save to file is in progress.[1]
Save to File[1]	When the connected signal transitions from False to True and Status is 0, the contents of the Application Log Definition instance is written to USB memory. Note that all application log data is written, regardless of access level.
Delimiter[1]	Delimiter used to separate the fields in the output *.CSV file.
File name[1]	Set the name of the file that is written when Save to File is activated. The length of the name without extension may not exceed eight characters, the name of the file extension may not exceed three characters.

[1] The item or status code is not supported by all HWD files.

Text Component Properties

Text and **Text List** components added to the **Screen Area Pane** provide the contents of the application log.



Application Data Logging

Text and Text List Properties

Item	Description
ActiveIndex[1]	Selects which text to use in the Text List.
Tag	This property determines which kind of Access Level the Access property will be compared to when reading the application log. It can be set to one of three values: • E = Error; see *Accessrights App Log Errors* on page 360. • S = Statistics; see *Accessrights App Log Statistics* on page 358. • O = Others; see *Accessrights App Log Others* on page 361.
Access	Sets the access level of the data. This value works with **Access** components to limit users' access to the application data log with the PLUS+1® Service Tool.
Write	Data will be written to the application data log. If Pending is True, the data will be queued until it can be written.
Pending	A write is ongoing.

[1] The item only exists in Text Lists.

Using Application Log 2

When applicable refer to *Vector-Based Screen Editor* on page 450.

Defining Texts

Text Definitions are defined in the Screen Library and can be used interchangeably in Application Log Definitions and Screen Definitions. Using the **Screen Library** popup menu, it is possible to arrange, create, delete and edit text definitions. For information on how to edit Text Definitions, see *Edit Text Window* on page 461.

Application Log Definition Interface

The interface of an Application Log Definition can be defined either in the Application Log 2 Editor or in the Write Applog component query dialog.

For information on how to manage the interface and how to connect interface signals to objects in the Vector Based Screen Editor, go to::

How to Manually Make Signals Available for Assignment to Screen Assets on page 471.

How to Assign an Available Signal to a Screen Asset on page 472.

How to Delete a Signal from a Screen Definition on page 473.

The Write Applog query dialog works analogously with the Show Screen query dialog.

For information on how to define interface signals in the Show Screen query dialog, go to:

About the Show Screen Component and the Query Screen Component Window on page 510.

About the Query Screen Component Window and Screen Definition Signals on page 511.

About the Query Screen Component Window and Screen Definition Buses on page 513.

Putting It Together

Writing text to the application log in runtime, can be done by following these steps.

Application Data Logging

1. Create a new Application Log Definition in the project manager or in the next step, then
2. Add a Write Applog component to the graphical drawing.
   a) Connect a True component to the ENABLE input and a bus to the IN input.
   b) Connect a boolean signal to the bus.
      This signal controls when the Application Log will be written.
3. Query the Write Applog component and create or select an Application Log Definition. Add the signal to the Application Log Definition.
4. Edit the Application Log Definition by double-clicking it in the Write Applog query dialog or the Project Manager.
   The **Application Log 2 Editor** will open.
5. Define a text in the Screen Library.
6. Add the text to the Screen Area Pane using drag and drop
7. Select the text in the Screen Manager or in the Screen Area Pane.
8. Connect the Boolean signal added to the interface bus to the write property of the text.

In runtime, when the write signal connected to the text is true, the text will be written to the application log.

Application Data Logging

How to Read the Contents of an Application Data Log

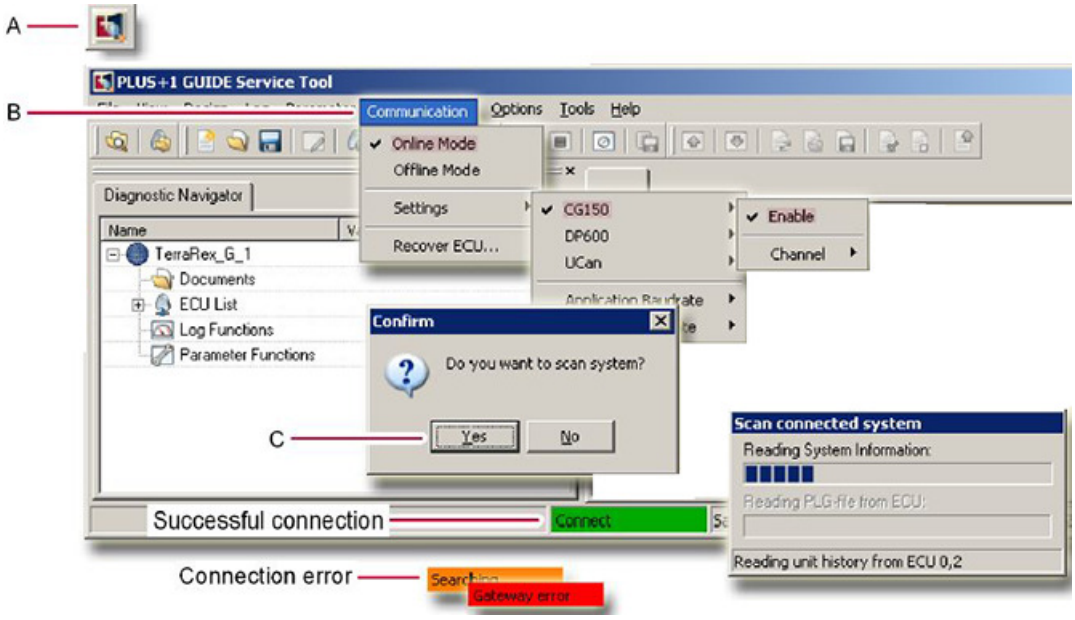
Use the PLUS+1® Service Tool program to access the contents of an application data log.

Warning

Without Tool Key protection, there is an increased risk that unauthorized personnel could use the PLUS +1® Service Tool program to view your application's data log and to change your application's operating parameters. Changes in your application's operating parameters could cause unexpected machine movements that result in personal injury and equipment damage. Always use the PLUS+1® GUIDE program's Tool Key feature to restrict access to your application's operating parameters. Tool Key protection reduces the risk that unauthorized personnel could view and change your application's operating parameters. Refer to *Use the Tool Key to Restrict Service Tool Access to Application Values*.

The following procedure describes accessing data from an application with a fully licensed version of the PLUS+1® Service Tool program, where **Access** components have not been used to restrict access to the contents of the application data log.

1. Set up the hardware and cabling needed for communication between the PLUS+1 controller and your PC.
2. Start the PLUS+1 Service Tool program.



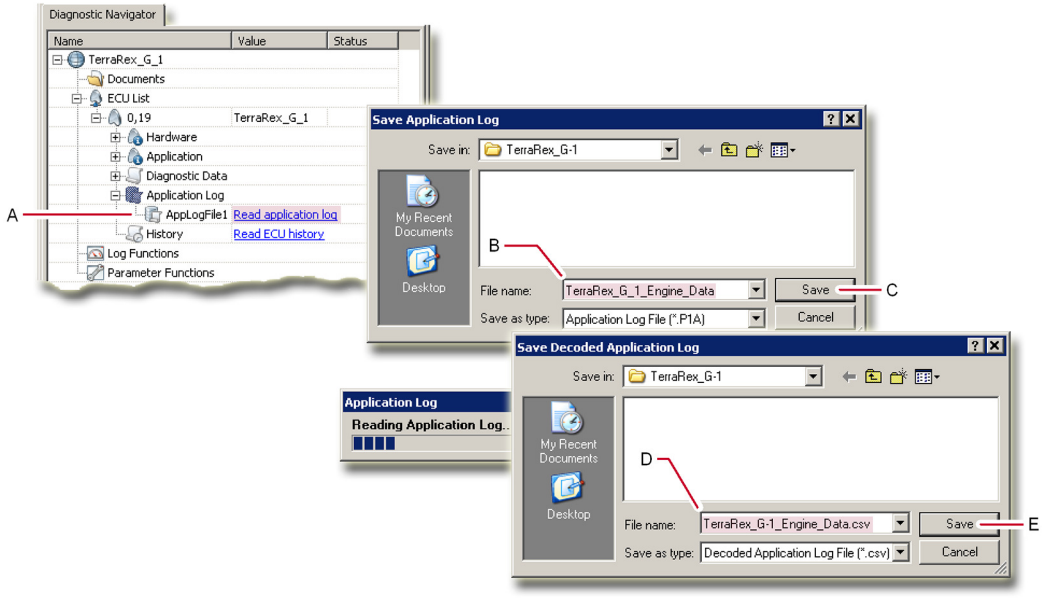
A Click the **PLUS+1 Service Tool** button in the **PLUS+1 GUIDE** window toolbar. You can also start this program through the **Start** menu on your PC.

B In the Service Tool program's **Communications** menu, check that the **Settings** command selects the **CG150** Communicator.

C If the **Confirm** window displays, click **Yes** to scan the hardware.

Application Data Logging

3. Write the contents of the application data log to CSV (comma separated value) file.



A In the **Diagnostic Navigator** tab's tree, click **Read application log**. The **Save Application Log** window opens.

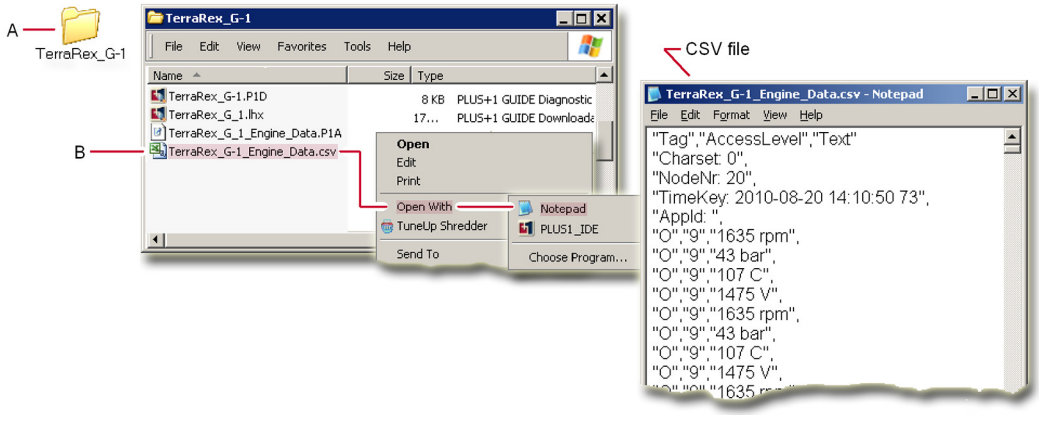
B In the **Save Application Log** window, enter the name of the application log file (*.P1A).

C In the **Save Application Log** window, click **Save** to write the contents of the application data log to the application log file. (The application log file (*.P1A) is an encrypted file. A users' ability to decode this file can be limited by using the **Access** property of the **Define Application Log Page** along with **Access** components.) The **Save Decoded Application Log** window displays.

D In the **Save Decoded Application Log** window, enter the **File name** for the CSV file.

E Click **Save**.

4. Open the CSV application log file.

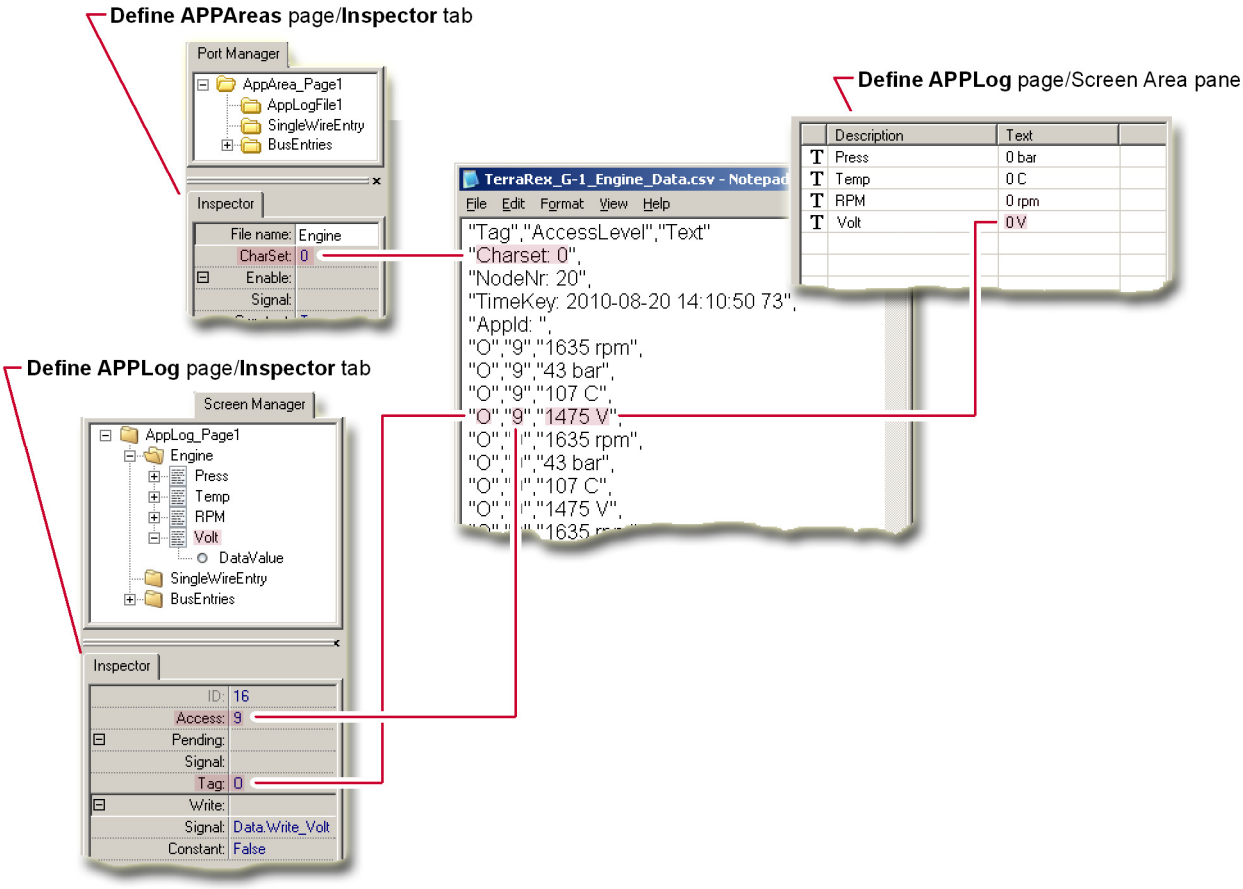


A Browse to the folder that contains the CSV application log file.

B Open the CSV application log file with Notepad.

Application Data Logging

About the Properties that Determine Data Logging Values



This figure shows some of the **Define Application Log Areas Page** and the **Define Application Log Page** properties that determine the display of data in the CSV application data log.

Support Tools

There are support tools for display of the contents of SCS files, selecting SCS files to compare, and running other very useful tasks.

Module Viewer

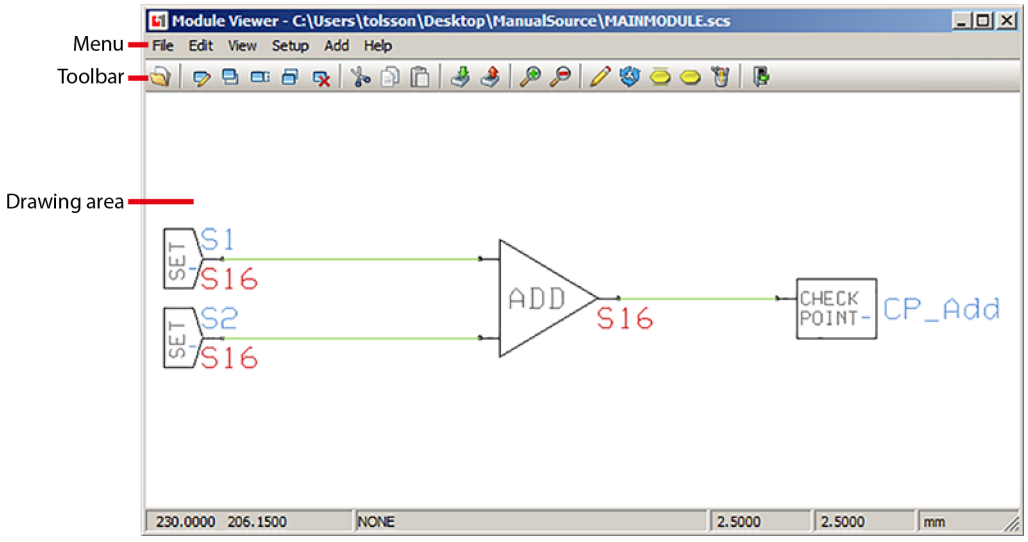
The **Module Viewer** is a tool that can be used to display the contents of SCS files.

It is possible to open any number of Module Viewers, either from GUIDE, or directly by double clicking on an SCS file.

The Module Viewer has the same editing possibilities as GUIDE itself, except for the Save operation. It is however possible to copy and paste freely between GUIDE and the**Module Viewer**.

Since each **Module Viewer** instance runs in its own processes, it is possible to open and close the **Module Viewer** independently from opening and closing GUIDE.

Module Viewer window



The **Module Viewer** window provides an additional window, independent of the main **PLUS+1 GUIDE** window, in which SCS files can be edited. Typical uses of this window include:

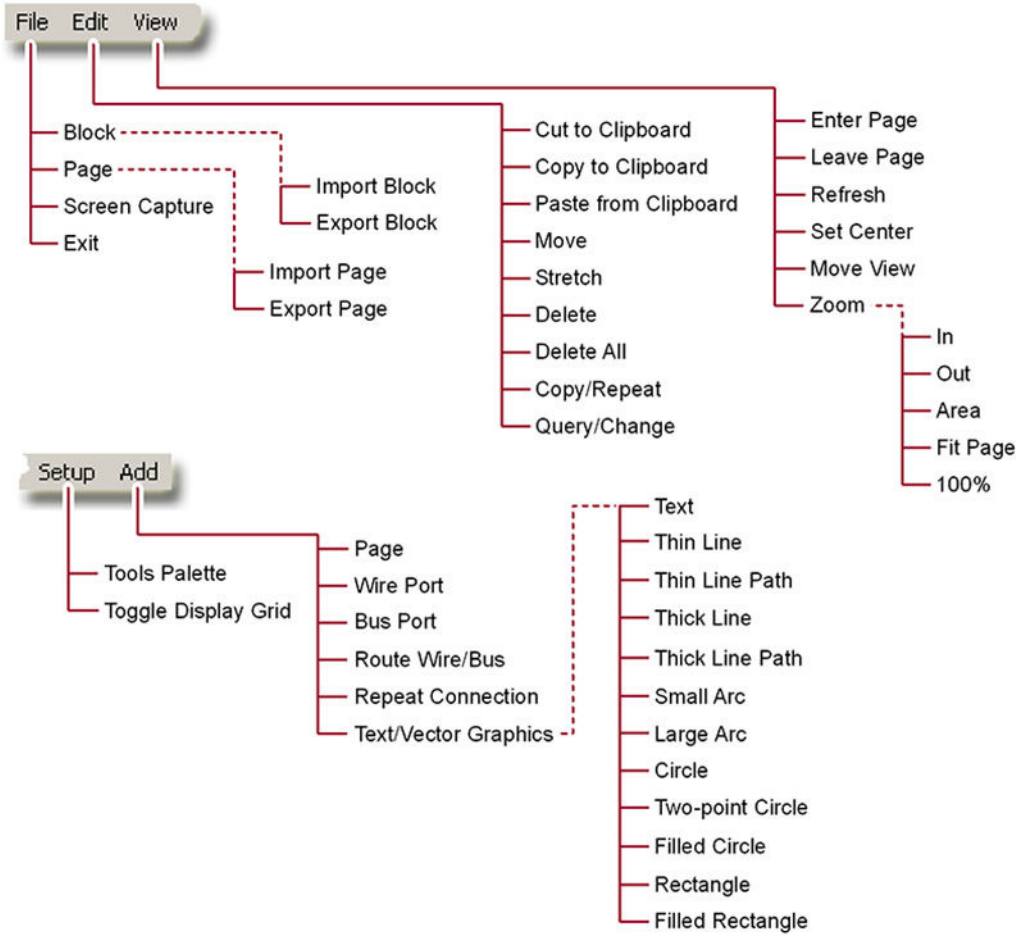
- Seeing multiple pages within the same project. (You can open the SCS file for your current project in this window).
- Coordinating CAN communications between two controllers.

Module Viewer window elements

Item	Description
Menu	Use the menu bar to access **Module Viewer** window commands.
Toolbar	Use the **Toolbar** to access commonly used **Module Viewer** window commands.
Drawing area	Edit the module here.

Support Tools

Module Viewer Window Menu



Use the menu bar to access commonly used **Module Viewer** window commands. See *Module Viewer Window Toolbar* on page 551 for more about this toolbar.

About — Help menu, which displays an information about the **Module Viewer** version and user license information.

File Menu in SCS Files

*File menu in the **Module Viewer** window's Drawing Area*

Item	Description
Open	Provides a File Open dialog to select another SCS file.
Save As	Saves the SCS file under a different file name.
Block	• **Import Block** displays the **PLUS1 Editor Load Block** window. Use this window to select and import an SCS file into the **Module Viewer** window's Drawing Area. When you import two or more identical blocks, the link feature automatically links the contents of the blocks. • **Export Block** displays the **Symbol Block Export Binary** window after you select items for export in the **Module Viewer** window's Drawing Area. Use this window to export the selected items to an SCS file. The `Import Block` and `Export Block` commands do not work with Define Areas and Define Screen blocks.

Support Tools

*File menu in the **Module Viewer** window's Drawing Area (continued)*

Item	Description
Page	• `Import Page` displays the **Select Job File Name** window after you select a page in the **Module Viewer** window's Drawing Area. Use this window to select an SCS file and import its contents into the selected page. • `Export Page` displays the **Select/Define New Job File Name** window after you select a page in the **Module Viewer** window's Drawing Area. The SCS file name becomes the page name when you import the page. The `Import Page` and `Export Page` commands do not work with Define Areas and Define Screen blocks.
Screen Capture	Displays the **Select Screen Dump Format** window after you select items in the **Module Viewer** window's Drawing Area. Use this window to choose to print the selected items, save them to a BMP or TIFF file, or copy them to the clipboard in a BMP or MTF format.
Close	Closes the **Module Viewer** window.

Edit Menu in SCS Files

*Edit menu in the **Module Viewer** window's Drawing Area*

Item	Description
Cut to Clipboard	Deletes items selected in the **Module Viewer** window's Drawing Area and copies them to the Clipboard.
Copy to Clipboard	Copies items selected in the **Module Viewer** window's Drawing Area to the Clipboard.
Paste from Clipboard	Pastes the contents of the Clipboard into the **Module Viewer** window's Drawing Area. The link feature automatically links the contents of duplicate pages that are copied from the Clipboard.
Move	Moves items selected in the **Module Viewer** window's Drawing Area.
Stretch	Stretches segments in selected routes and moves selected items. Click a route to add a vertex to the route. The new vertex can then be selected and moved without changing the positions of other vertexes in the route.
Delete	Deletes items selected in the **Module Viewer** window's Drawing Area. Selected items turn white. • The **Attributes** window displays when you select a single item or identical items. Click `OK` to delete your selection. • The **Select Item Class** window displays when you select different items. Use this window to select the items that you want to delete. Click `OK` to delete your selections. — Asterisks (*) identify items that will be deleted. — Dashes (–) identify items that will not be deleted.
Delete All	Displays a **Question** window with a **Delete all items?** message. Click **Yes** to delete all items in the **Module Viewer** window's Drawing Area.
Copy/Repeat	Copies items selected in the **Module Viewer** window's Drawing Area. Click to place copied items. Press **ESC** to stop placing copied items. This command only works within the Drawing Area. The link feature automatically links copied pages.
Query/Change	Use to change the properties (such as data type, text, and function) of items in the **Module Viewer** window's Drawing Area. Click an item whose property you want to change. A dialog box appropriate for the selected item displays. Change the property in this window.

View Menu in SCS Files

*View menu in the **Module Viewer** window's Drawing Area*

Item	Description
Enter Page	Enters a selected page. To enter a page, click within the page boundaries or drag at a page port.
Leave Page	Leaves the current page.
Refresh	Refreshes the **Module Viewer** window's Drawing Area.
Set Center	Centers the **Module Viewer** window's Drawing Area on where you click the pointer.
Move View	Displays a movable, transparent rectangle with white borders. Click to center the **Module Viewer** window's Drawing Area within the borders of this rectangle.

Support Tools

*View menu in the **Module Viewer** window's Drawing Area (continued)*

Item	Description
Zoom	• **In** zooms in and centers the**Module Viewer** window's Drawing Area on where you click the pointer. • **Out** zooms out and centers the **Module Viewer** window's Drawing Area on where you click the pointer. • **Area** zooms the **Module Viewer** window's Drawing Area on an area defined by two pointer clicks. • **Fit Page** sizes the view to fit all items on the page into the **Module Viewer** window's Drawing Area. • **100%** zooms the view to midway between the minimum and maximum zooms.
Compare SCS Files…	Starts up the **Compare SCS Files…** tool to allow comparing the current SCS file with another SCS file.
Reload	Reloads the Module SCS file from disk.

Setup Menu in SCS Files

*Setup menu in the **Module Viewer** window's Drawing Area*

Item	Description
Toggle Display Grid	Turns the display grid off and on. The display grid is the grid that you see in the Drawing Areas of the PLUS+1® GUIDE, **Page Interface Editor**, and **Module Viewer** windows.
Tools Palette	Displays the **Icon Menu** window.
Options...	Displays the options dialog.

Add Menu in SCS Files

*Add menu in the **Module Viewer** window's Drawing Area*

Item	Description
Page	Adds a page where you click and drag in the **Module Viewer** window's Drawing Area.
Wire Port	Displays the **Select Signal–Schematics Design** window. Use this window to add a Wire Port to the **Module Viewer** window's Drawing Area. **1.** Enter the port name in **EntryName** field. **2.** Click OK. **3.** Click to place the port in the **Module Viewer** window's Drawing Area.
Bus Port	Displays the **Select Signal–Schematics Design** window. Use this window to add a Bus Port to the **Module Viewer** window's Drawing Area. **1.** Enter the port name in **EntryName** field. **2.** Click OK. **3.** Click to place the port in the **Module Viewer** window's Drawing Area.
Route Wire/Bus	Starts drawing either a green single signal wire or a red multi-signal bus from where you click in the **Module Viewer** window's Drawing Area. You get: • a bus when you start routing from a bus source. • a wire when you start routing from a wire source. K toggles the route between a wire and a bus. F9 terminates unconnected routes.
Repeat Connection	Duplicates a selected route connection in the **Module Viewer** window's Drawing Area. Repeated connections automatically have a number added to their name: *Signal_Name*, *Signal_Name2*, *Signal_Name3*, *Signal_Name4*.
Text/Vector Graphics	Displays a menu of text and graphics commands. Use these commands to add text and graphic elements to the **Module Viewer** window's Drawing Area.

Support Tools

Module Viewer Window Toolbar

Buttons on the Module Viewer window toolbar access commonly used Modular Viewer commands.

See *Module Viewer Window Menu* on page 548 for a list of all commands available through the menu bar.

Descriptions of buttons on the Module Viewer window toolbar

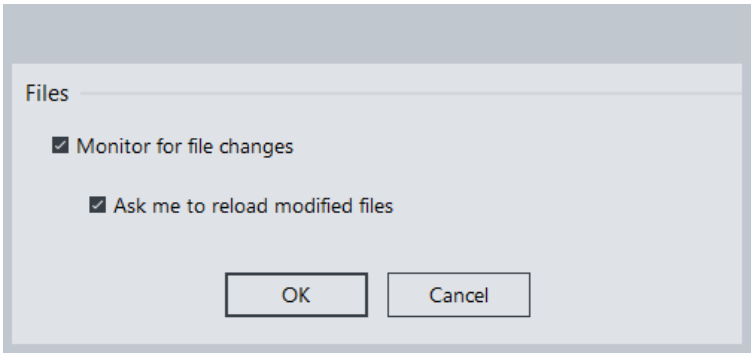
Item		Description
	Query/Change	Use to change the properties (such as data type, text, and function) of items in the drawing area of the **Module Viewer** window. Click an item whose property you want to change. A dialog box appropriate for the selected item displays. Change the property in this window.
	Move	Moves items selected in the drawing area of the **Module Viewer** window.
	Stretch	Stretches segments in selected routes and moves selected items. Click a route to add a vertex to the route. The new vertex can then be selected and moved without changing the positions of other vertexes in the route.
	Copy/Repeat	Copies items selected in the drawing area of the **Module Viewer** window. Click to place copied items. Press **ESC** to stop placing copied items. This command only works in the drawing area. The link feature automatically links copied pages.
	Delete	Deletes items selected in the drawing area of the **Module Viewer** window. Selected items turn white. • The **Attributes** window displays when you select a single item or identical items. Click **OK** to delete your selection. • The **Select Item Class** window displays when you select different items. Use this window to select the items that you want to delete. Click **OK** to delete your selections. — Asterisks (*) identify items that will be deleted. — Dashes (–) identify items that will not be deleted.
	Cut to Clipboard	Deletes items selected in the drawing area of the **Module Viewer** window and copies them to the clipboard.
	Copy to Clipboard	Copies items selected in the drawing area of the **Module Viewer** window to the clipboard.
	Paste from Clipboard	Pastes the contents of the clipboard into the drawing area of the **Module Viewer** window. The link feature automatically links the contents of pages that are copied from the clipboard.
	Enter Page	Enters a selected page. To enter a page, click within the page boundaries or drag at a page port.
	Leave Page	Leaves the current page.
	Zoom In	Zooms in and centers the drawing on the area of the **Module Viewer** window on where you click the pointer.
	Zoom Out	Zooms out and centers the drawing area of the **Module Viewer** window on where you click the pointer.
	Route Wire/Bus	Starts drawing either a green single signal wire or a red multi-signal bus from where you click in the drawing area. The drawing will be either: • a bus when you start routing from a bus source. • a wire when you start routing from a wire source. K toggles the route between a wire and a bus. F9 terminates unconnected routes.
	Repeat Connection	Duplicates a selected route connection in the drawing area of the . Repeated connections automatically have a number added to their name: *Signal_Name*, *Signal_Name2*, *Signal_Name3*, *Signal_Name4*.
	Port Bus	Displays the window. Use this window to add a Bus Port to the drawing area: **1.** Enter the port name in field. **2.** Click **OK**. **3.** Click to place the port in the drawing area.

Support Tools

Descriptions of buttons on the Module Viewer window toolbar (continued)

Item		Description
	Wire Port	Displays the window. Use this window to add a Wire Port to the drawing area: **1.** Enter the port name in **EntryName** field. **2.** Click **OK**. **3.** Click to place the port in the drawing area.
	Drawing Tools	Displays a drop-down list of text and basic graphics tools. Use these tools to add text and vector graphics elements to the drawing area.
	Close	Closes the window.

Module viewer options



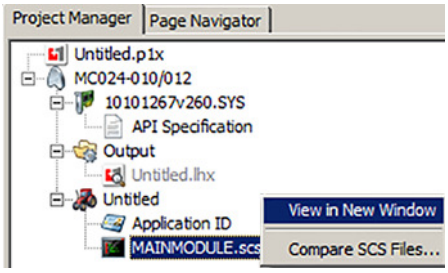
Item	Description
Monitor for file changes	If the SCS file is modified on disk, this setting allows the user to be notified about this. If checked, an asterisk is displayed next to the modified file's name.
Ask me to reload modified files	When both the previous and this option are checked, a yes/no reload dialog will be displayed when the SCS file is modified. (Files can also be reloaded using the Reload menu, or **F5** key.)

Support Tools

Starting the Module Viewer

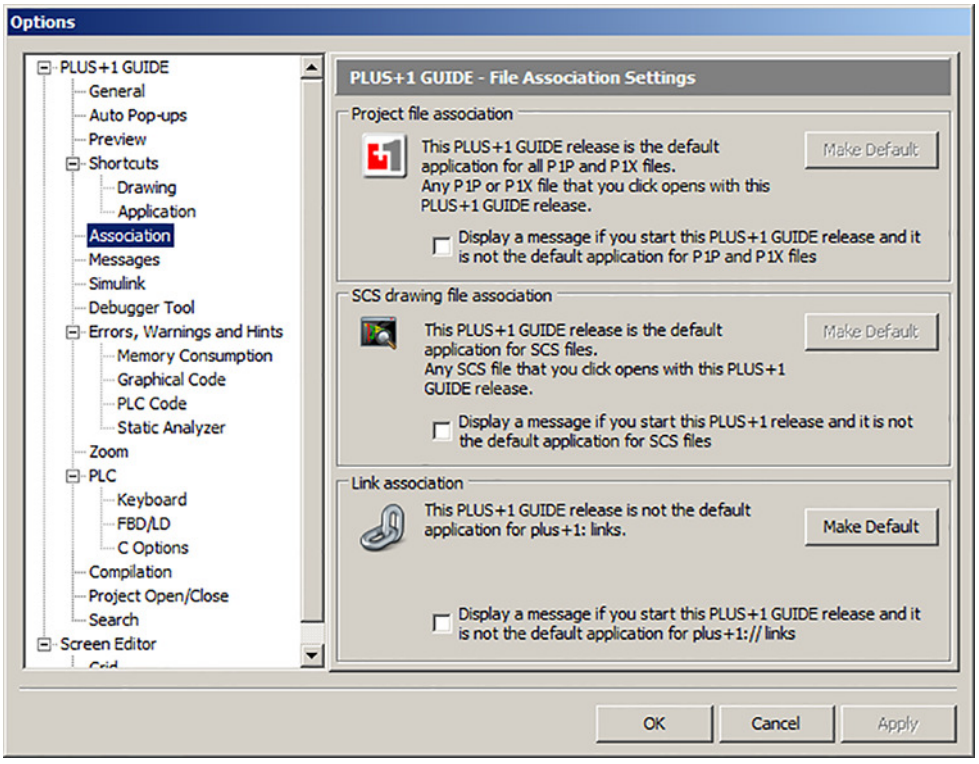
Starting the Module Viewer from GUIDE

The module viewer can be started from GUIDE by using the context menu item View in New Window.



Starting the Module Viewer from file association

The Module Viewer can also be opened by double clicking on an SCS file, provided that the File Association for SCS files has been made in GUIDE options.



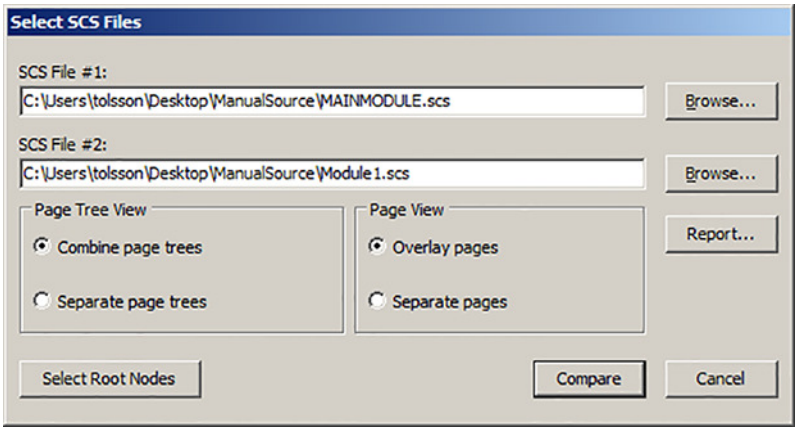
Starting the Module Viewer from the command line

The Module Viewer EXE is placed in the P1Tools folder and named ModuleViewer.exe. It takes one parameter; the name of the SCS file to open.

Support Tools

Compare SCS Files

Tools menu > **Select SCS Files** window:



Use **Select SCS Files** window to:

- Select two SCS files that have pages that you want to compare.
- Output a comparison report on the pages in the selected SCS files.
- Configure the initial display of a PLUS+1® **Compare SCS** window. (You explore differences between pages in the PLUS+1® **Compare SCS** window.)
- Begin a new comparison that starts from previously selected comparison pages.

This is a PLUS+1® GUIDE upgrade feature. A Quality Assurance License (reference PLUS+1® GUIDE add on license Quality Assurance Data Sheet, **AI170686484256**) enables this feature. For more information, see PLUS+1® GUIDE Licensing, **AQ419969404812**.

This feature is not licensed for use in all regions as described in the End User License Agreement.

Select SCS Files description

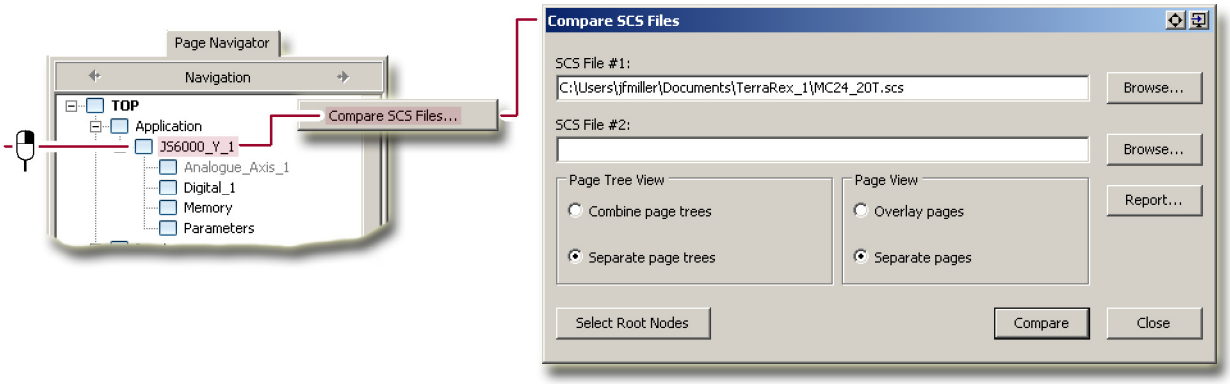
Item	Description
SCS File #1	This field shows the path to the first SCS file that you select for comparison. The **Compare** button in this window opens a PLUS+1® **Compare SCS** window. The PLUS+1® **Compare SCS** window compares pages in the SCS file shown in this field with pages in the SCS file shown in the **SCS File #2** field. • **Browse** opens an **Open** window. Use this window to browse to and select the first SCS file for comparison.
SCS File #2	This field shows the path to the second SCS file that you select for comparison. The **Compare** button in this window opens a PLUS+1® **Compare SCS** window. The **PLUS+1 Compare SCS** window compares the pages in the SCS file shown in this field with pages in the SCS file shown in the **SCS File #1** field. • **Browse** opens an **Open** window. Use this window to browse to and select the second SCS file for comparison.
Report	Opens a **Save As** window. Use this window to save a file in XML format that reports the similarities and differences between the pages of the selected SCS files.
Page Tree View	This option sets how the PLUS+1® **Compare SCS** window shows page tree views of the SCS files. • **Combine page trees** shows a view that combines the pages from both SCS files into a single page tree. • **Separate page trees** shows a separate page tree view for each SCS file.
Page View	This option sets how the PLUS+1® **Compare SCS** window displays the comparison pages. • **Overlay pages** overlays page views of the comparison pages. • **Separate pages** shows separate page views of the comparison pages.
Select Root Nodes	• Opens a **Select Root Nodes** window. • This window shows separate tree views of **SCS File#1** and **SCS File #2**. • Click pages in these tree views to set the starting pages ("root nodes") from which a comparison starts.
Compare	Opens the PLUS+1® **Compare SCS** window. Use this window to explore the differences between the pages in the two comparison files.

The **Compare SCS Files** feature can only find differences in SCS files that have been saved. Always save your changes before you compare SCS files.

Support Tools

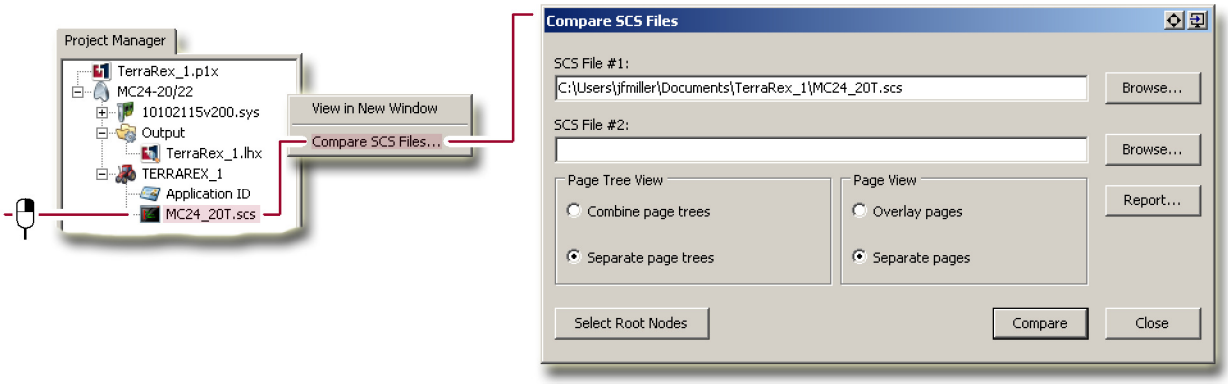
Starting PLUS+1 Compare SCS from the Page Navigator Tab

Right-click on a page in the **Page Navigator** tab to open the **Compare SCS Files** window.



Starting PLUS+1 Compare SCS from the Project Manager Tab

Right-click the SCS file in the **Project Navigator** tab to open the **Compare SCS Files** window.



Support Tools

Starting PLUS+1 Compare SCS from the Command Line

The Compare SCS EXE is placed in the P1Tools folder and named `CompareSCS.exe`

Compare SCS EXE required parameters

Parameter	Description
<SCS file names>	Up to two SCS file names can be provided to Compare SCS. • When given zero SCS file names, Compare SCS will work like previously, both SCS files are selected from previously used SCS files (if any). • When given one SCS-file, this SCS file will be used as the first SCS file. In this case, the field used for the second SCS file will be cleared empty. • When given two SCS files, they are used as the first and second SCS files. • When given more than two SCS files, all extra SCS files are ignored.
-auto	Only applicable when there are two selected SCS files, either from the command line or from previously used SCS files. The compare window is opened automatically for these two files.
-combine-page-trees	Combine page trees will be selected. If combined with -separate-page-trees, then the last one on the command line will win.
-separate-page-trees	Separate page trees will be selected. If combined with -combine-page-trees, then the last one on the command line will win.
-overlay-pages	Overlay pages will be selected. If combined with -separate-pages, then the last one on the command line will win.
-separate-pages	Separate pages will be selected. If combined with -overlay-pages, then the last one on the command line will win.
-pixel-comparison-view	Pixel comparison view will be used. If combined with -checksum-comparison-view, then the last one on the command line will win. Only applicable in combination with -overlay-pages.
-checksum-comparison-view	Checksum comparison view will be used. If combined with -pixel-comparison-view, then the last one on the command line will win. Only applicable in combination with -overlay-pages.
-left-root-path <page path>	Used to define root node for the left SCS file.
-right-root-path <page path>	Used to define root node for the right SCS file.

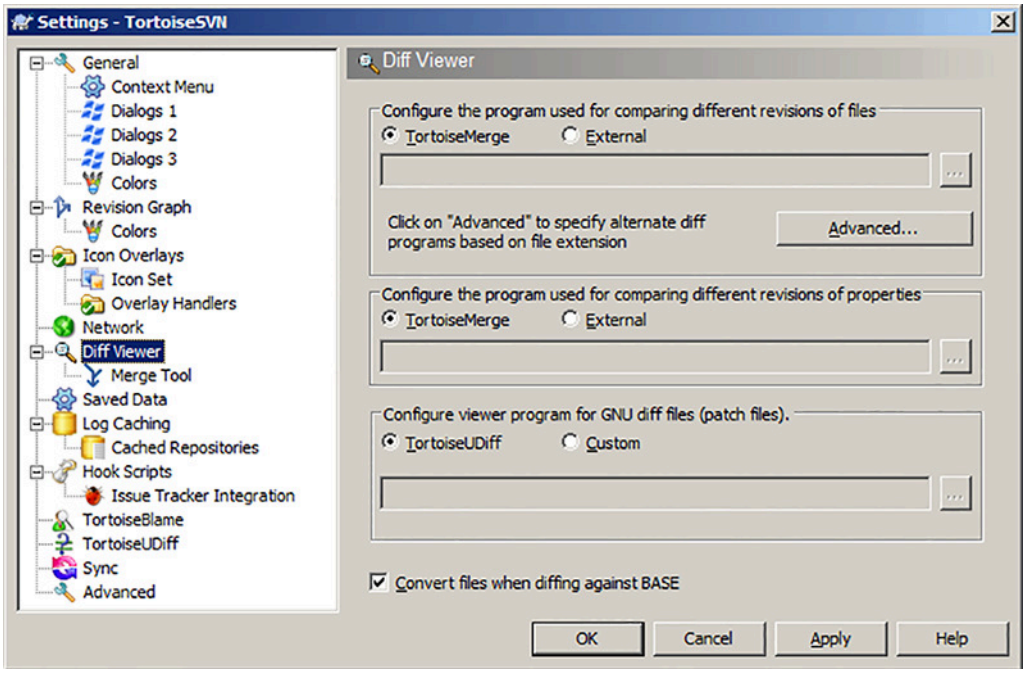
Support Tools

How to Add PLUS+1 Compare SCS as a Diff Tool to TortoiseSVN

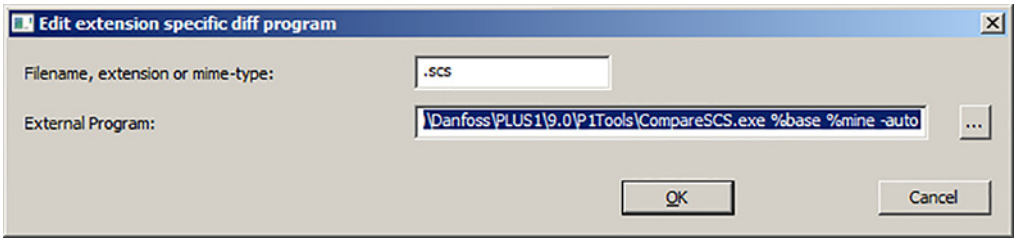
PLUS+1® Compare SCS can be used as a diff tool integrated with TortoiseSVN, TortoiseGit and other tools.

The following applies to `TortoiseSVN 1.9.3`

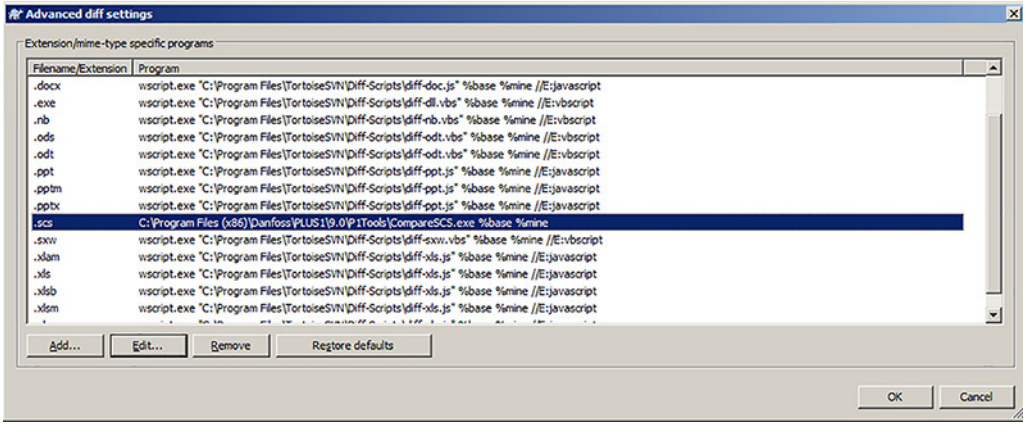
1. Select **Advanced** under **SVN Settings** > **Diff Viewer**.



2. **Edit extension specific diff program** type in extension `.scs` and external program `C:\Program Files (x86)\Danfoss\PLUS1\9.0\P1Tools\CompareSCS.exe %base %mine -auto`.



Handler for .scs files is selected to add.



%base is the first SCS file and *%mine* is the second SCS file.

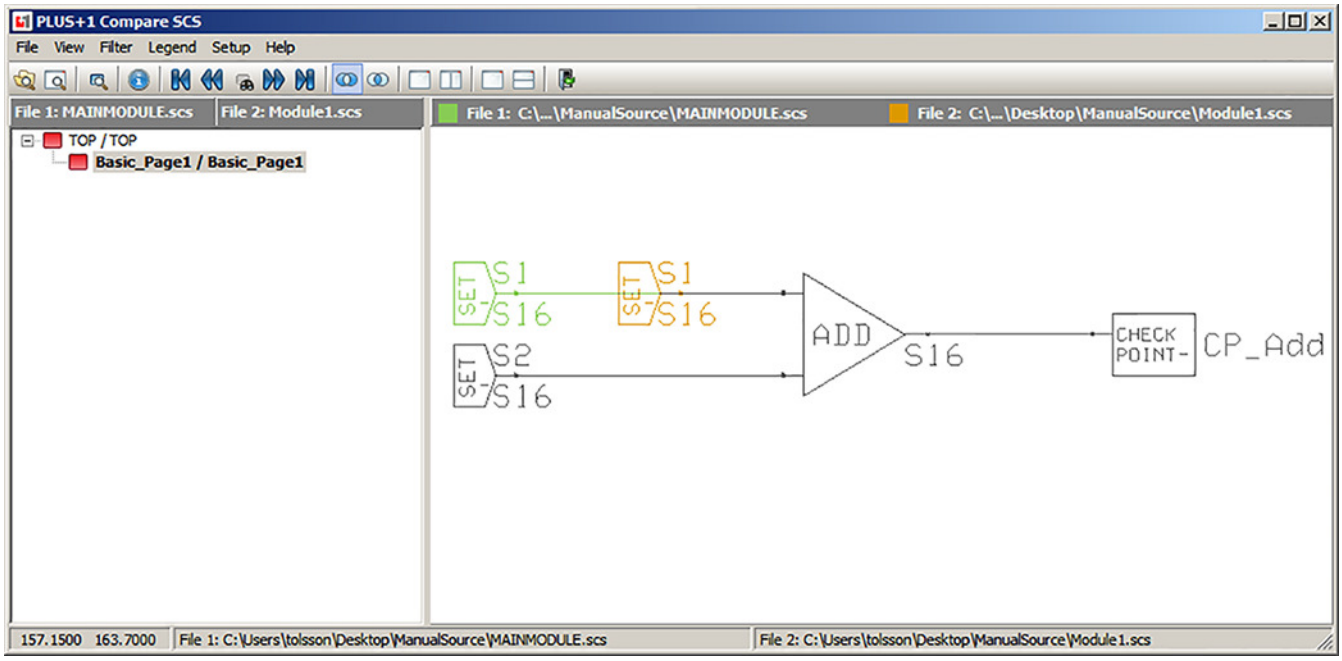
The parameter, -auto, makes the file comparison view open immediately without further user interaction.

Support Tools

PLUS+1 Compare SCS

Use this window to explore the differences between the pages of the two SCS files that you selected for comparison in the **Select SCS Files** window.

You can use the PLUS+1 GUIDE program's standard zoom, center, and move center tools in the **PLUS+1 Compare SCS** window's Page View tab (item **2**).



PLUS+1 Compare SCS description

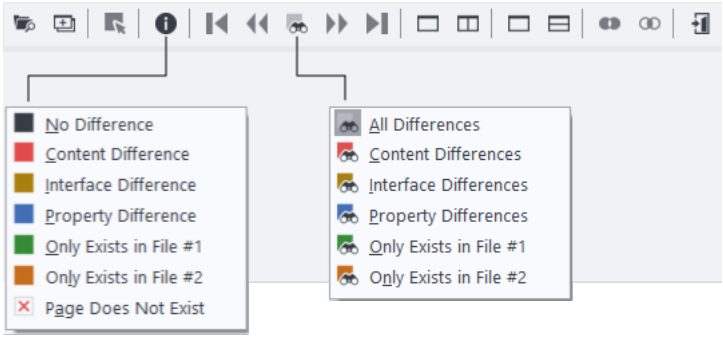
Item		Description
1.	**Page Tree View** tab	The **Select SCS Files** window has **Combine page trees** and **Separate page trees** options that set the type of page tree view that appears in the **PLUS+1 Compare SCS** window. If you select the: • **Combine page trees** option, the **PLUS+1 Compare SCS** window opens with a single page tree view. Pages from SCS File #1 are on the left side of the page tree view and pages from SCS File #2 are on the right side of the page tree view. • **Separate page trees** option, the PLUS+1 Compare SCS opens with two separate page tree views, as shown in the preceding figure. The page tree view for SCS File #1 is on the left. The page tree view for SCS File #2 is on the right. Brightly colored page icons in the page tree views indicate the type of difference between pages. To change the page tree view, return to the **PLUS+1 Compare SCS** window and select a different page tree view option. *When you left-click a page* in the page tree view, the Compare SCS files feature: • Selects the best matching page from the other SCS file. • Displays the selected pages in the Page View tab. *When you right-click a page* in the page tree view, the Compare SCS files feature: • Opens the **Select Root Nodes** window, which displays separate tree views of the two comparison SCS files. Click pages in these tree views to set the starting pages (root nodes) from which a comparison starts.
2.	**Page View** tab	Shows either separate page views or overlaid page views of the pages that you select in the **Page Tree View** tab.
3.	**Toolbar**	See *PLUS+1 Compare SCS Toolbar* on page 559 for description of using the controls in this toolbar.

Support Tools

PLUS+1 Compare SCS Toolbar

Use the controls in the **Compare Pages** window toolbar to:

- return to the **Compare SCS** window.
- open the **Select Root Nodes** window, where you can set the starting pages ("root nodes") from which a comparison starts.
- find differences between pages.
- switch between a single overlaid page view and two separate page views of the selected comparison pages.
- view the checksum or pixel position differences between the selected comparison pages.



PLUS+1 Compare SCS toolbar

Icon	Item	Description
	New Compare	Opens the Select SCS Files dialog. Use this to make a new Comparison between SCS files.
	New Compare window	Opens a new instance of PLUS+1 Compare SCS. The currently open comparison will remain intact.
	Select Root Nodes	Opens the **Select Root Nodes** window. This window displays separate tree views of the two comparison SCS files. Click pages in these tree views to set the starting pages ("root nodes") from which a comparison starts.
	Legend	The colors of the page icons indicate the type of difference between pages. For more details see the following *Legend button details* table.
	First Difference	Go back to the first different pages in the page tree view. Highlight the page names and display page views of these pages.
	Previous Difference	Go back to the previous different pages in the page tree view. Highlight the page names and display page views of these pages.
	Find	Apply search refinements to the Difference buttons. For more details see the following *Legend button details* table.
	Next Difference	Go forward to the next different pages in the page tree view. Highlight the page names and display page views of these pages.
	Last Difference	Go back to the last different pages in the page tree view. Highlight the page names and display page views of these pages.
	Pixel Comparison View	Select the Page Overlay View to make this button available. Click this button to make an XOR-type comparison between pixel positions in the**File 1** and **File 2** pages. If a pixel position in the: • **File 1** page matches a pixel position in the **File 2** page, then both pixels gray out. • **File 1** page does not match a pixel position in the **File 1** page, then the unmatched pixel turns green. • **File 2** page does not match a pixel position in the **File 1** page, then the unmatched pixel turns brown.
	Checksum Comparison View	Select the One View (page overlay) view to make this button available. Click this button to make an XOR-type comparison between checksums in the **File 1** and **File 2** pages. If a checksum for an item in the: • **File 1** page matches the checksum for an item in the **File 2** page, then both items gray out. • **File 1** page does not match the checksum for an item in the **File 2** page, then the unmatched item turns green. • **File 2** page does not match the checksum for an item in the **File 1** page, then the unmatched item turns brown.

Support Tools

PLUS+1 Compare SCS toolbar (continued)

Icon	Item	Description
	Overlay Page View	The Page View tab overlays page views of the selected pages. This view enables the Pixel Comparison View and the Checksum Comparison View buttons. In this view, click the: • **Pixel Comparison View** button to view pixel position differences between the two pages. • **Checksum Comparison View** button to view checksum differences between the two pages.
	Separate Page Views	The Page View pane shows separate page views of the selected pages. This view disables the Pixel Comparison View button and the Checksum Comparison View buttons. Use this view to see checksum differences between the two pages.
	Exit	Closes down this instance of Compare SCS.

Legend button details

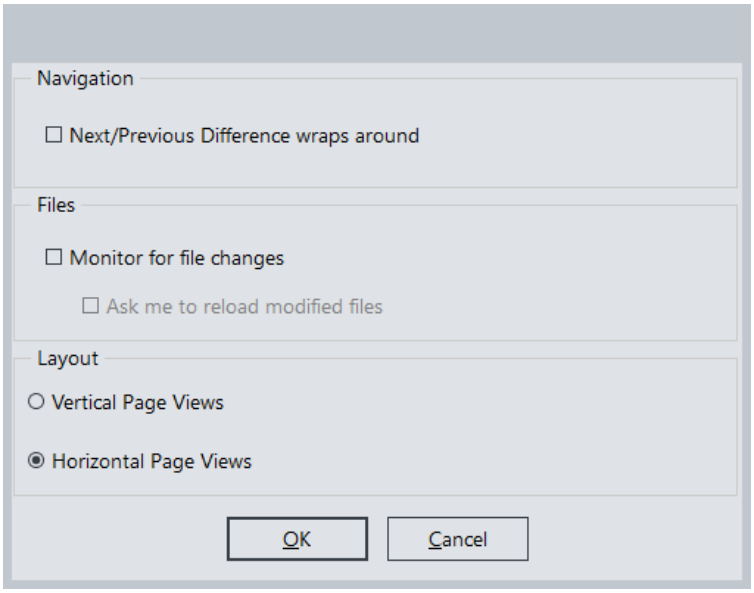
Icon	Description
	No Difference—the pages are identical.
	Content Difference—the pages have a content-related checksum difference.
	Interface Difference—the pages have an interface-related checksum difference.
	Property Difference—the pages have a property-related checksum difference.
	Only Exists in File #1—the **File 1** page has a checksum for an item that does not exist in the **File 2** page.
	Only Exists in File #2—**the File 2** page has a checksum for an item that does not exist in the **File 1** page.
	Page Does Not Exist—a page from which a reasonable comparison can be made does not exist.

Find button details

Icon	Description
	All Differences—the Difference buttons take you from one different page to the next different page.
	Content Differences—the Difference buttons take you only to pages whose content-related checksums differ.
	Interface Differences—the Difference buttons take you only to pages whose interface-related checksums differ.
	Property Differences—the Difference buttons take you only to pages whose property-related checksums differ.
	Only Exists in File 1—the Difference buttons take you only to **File 1** pages that have checksums for items that do not exist in **File 2** pages.
	Only Exists in File 2—the Difference buttons take you only to **File 2** pages that have checksums for items that do not exist in **File 1** pages.

Support Tools

Compare SCS options



Item	Descriptions
Next/Previous Difference wraps around	This setting controls whether the toolbar buttons for Next/Previous difference wraps around or not.
Monitor for file changes	If either of the compared SCS files is modified on disk, this setting allows the user to be notified about this. If checked, an asterisk is displayed next to the modified file's name.
Ask me to reload modified files	When both the previous and this option are checked, a yes/no reload dialog will be displayed when an SCS file is modified. (Files can also be reloaded using the Reload menu, or **F5** key.)
Vertical Page Views	When selected, the SCS files are displayed with File1 over File2.
Horizontal Page Views	When selected, the SCS files are displayed with File1 to the left of File2.

About the Order in which Checksum Differences are Identified

When comparison pages have more than one type of checksum difference, differences are shown in the following order:

1. – Content-related checksum difference.
2. – Interface-related checksum difference.
3. – Property-related checksum difference.

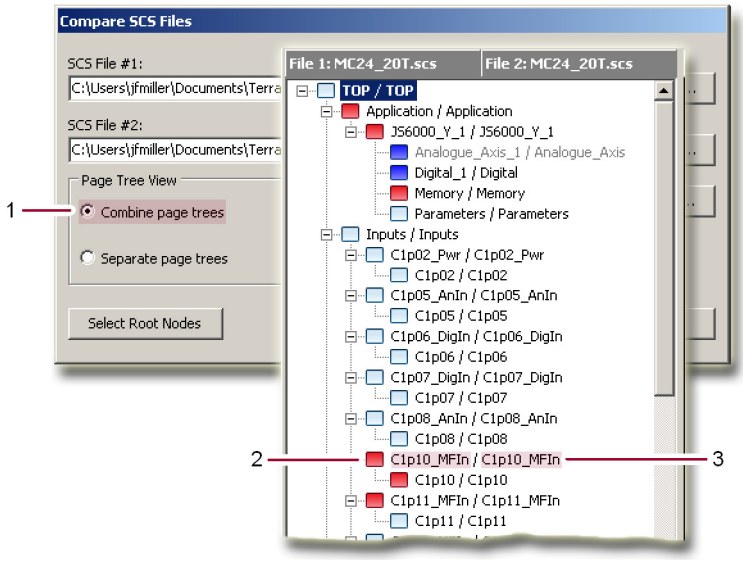
About the Page Tree View

The **Select SCS Files** window's **Page Tree View** option configures the **PLUS+1 Compare SCS** window to display either:

Support Tools

- A **Combine page tree**s view that combines pages from SCS File #1 and SCS File #2 into a single page tree.
- Or a **Separate Page trees** view that separates pages from SCS File #1 and SCS File #2 into two separate page trees.

About the Single, Combined Page Tree View



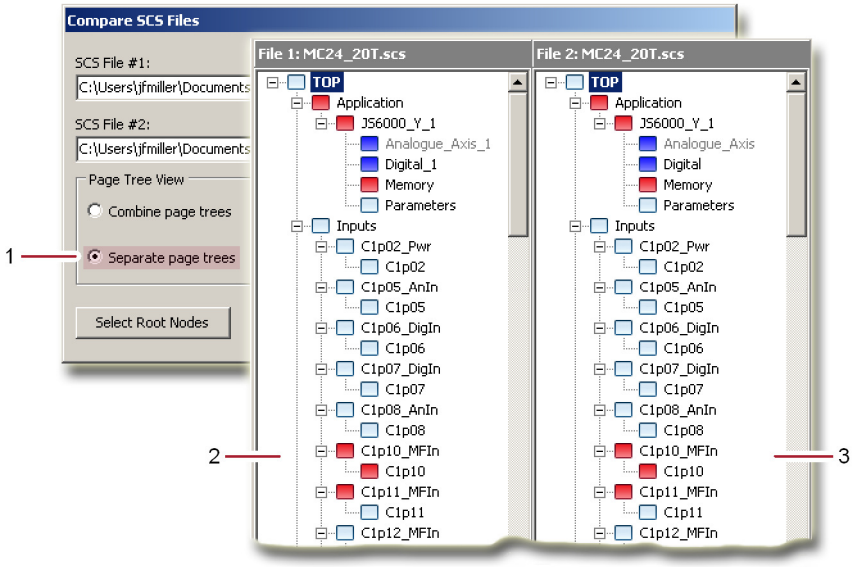
1. If you select the **Combine page trees** option, the **Compare Pages** window displays separate page tree views for **SCS File #1** and **SCS File #2**.
2. Pages from **SCS File #1** are in the left side of page tree view.
3. Pages from **SCS File #2** are in the right side of page tree view.

Text color indicates the Page View Access:

Black	= Normal
Green	= Force Enabled
Maroon	= Read Only
Gray	= Disabled

Support Tools

About the Separate Page-Tree View



1. If you select the **Separate page trees** option, the **Compare Pages** window displays separate page tree views for **SCS File #1** and **SCS File #2**.
2. Pages from **SCS File #1** are in the left page tree view.
3. Pages from **SCS File #2** are in the right page tree view.

Text color indicates the Page View Access:

Black	= Normal
Green	= Force Enabled
Maroon	= Read Only
Gray	= Disabled

About the Overlay Pages and the Separate Pages Views

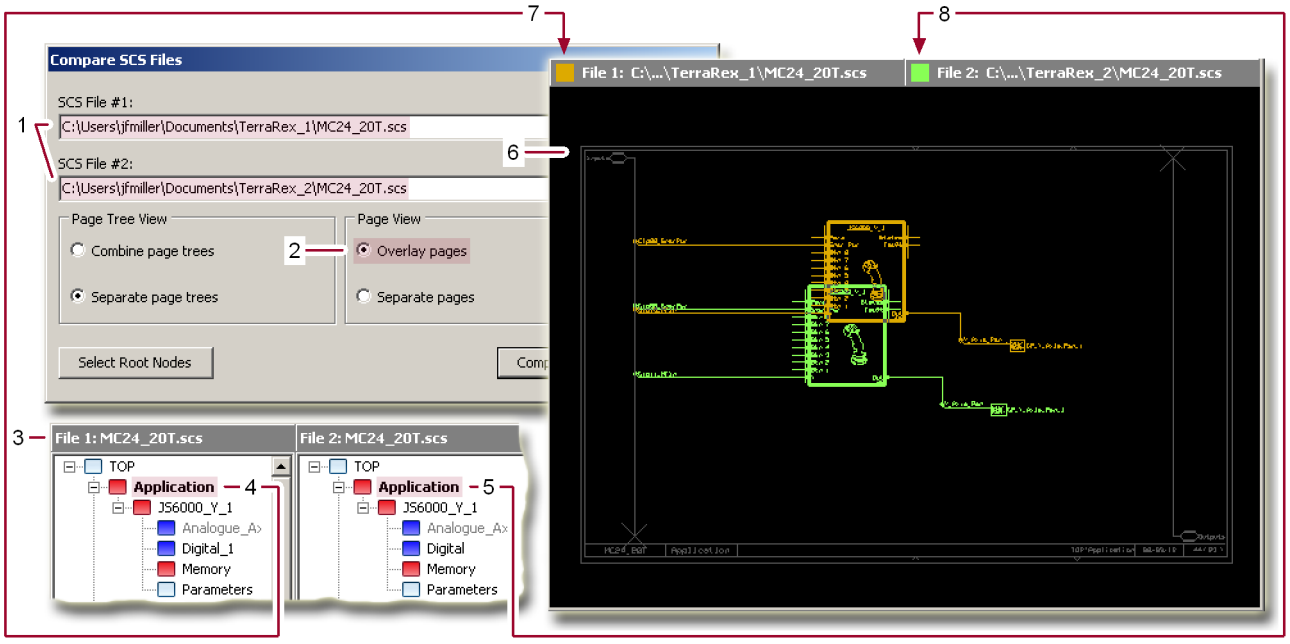
The **Compare SCS File** window's **Page View** options configure the **Compare Pages** window to display either:

- An **Overlay pages** view that overlays the comparison pages into one page view.
- Or a **Separate pages** view that separates the comparison pages into two page views.

You can use the Overlay Page View and the Separate Page View buttons in the **Compare Pages** window to switch between overlaid and separate page views.

Support Tools

About the Overlay Pages View

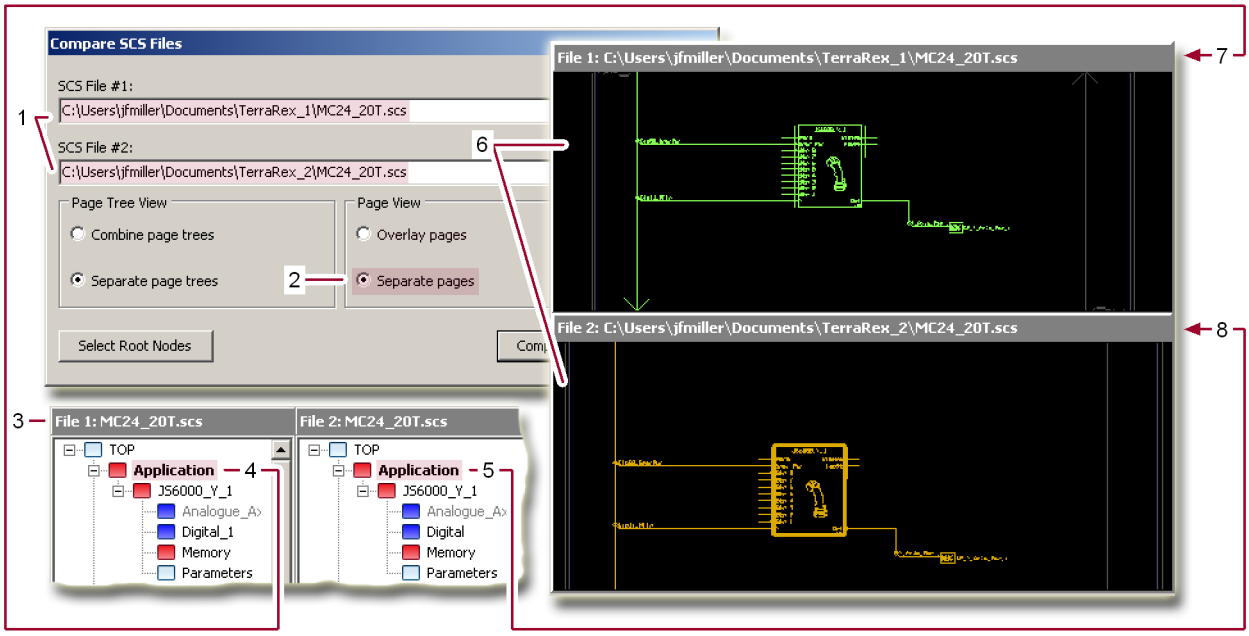


Overlay Pages view description

Item	Description
1.	The **Compare SCS Files** window sets the paths to the two comparison SCS files.
2.	Selecting the **Overlay pages** option in the **Compare SCS Files** window causes the **Compare Pages** window to overlay page views of the comparison pages.
3.	The **Compare Pages** window's page tree views. Select pages for comparison here.
4.	The **File 1** page selected in the **Compare Pages** window for viewing in this window's Page View pane.
5.	The **File 2**page selected in the **Compare Pages** window for viewing in this window's Page View pane.
6.	The **Compare Pages** window's Page View pane overlays page views of the **File 1** and **File 2** pages.
7.	The **Compare Pages** window's Page View pane. Brown highlights where the **File 1** page differs from the **File 2** page. Use buttons in the **Compare Pages** window's toolbar to see differences in either pixel positions or checksum values. Grayed-out items have matching pixel positions or checksum values in both the **File 1** and **File 2** pages.
8.	The **Compare Pages** window's Page View pane. Green highlights where the **File 2** page differs from the **File 1** page. Use buttons in the **Compare Pages** window's toolbar to see differences in either pixel positions or checksum values. Grayed-out items have matching pixel positions or checksum values in both the **File 1** and **File 2** pages.

Support Tools

About the Separate Pages View



Separate Pages view description

Item	Description
1.	The **Compare SCS Files** window sets the paths to the two comparison SCS files.
2.	Selecting the **Separate pages** option in the **Compare SCS Files** window causes the **Compare Pages** window to show separate page views of the comparison pages.
3.	The **Compare Pages** window's page tree views. Select pages for comparison here.
4.	The **File 1** page selected in the **Compare Pages** window for viewing in this window's upper Page View pane.
5.	The **File 2**page selected in the **Compare Pages** window for viewing in this window's lower Page View pane.
6.	The **Compare Pages** window's Page View panes show separate page views of the **File 1** and **File 2** pages.
7.	The **Compare Pages** window's Page View pane. Green highlights where checksum values differ in the **File 1** page from checksum values in the **File 2** page. Grayed-out items have matching checksum values in both the **File 1** and **File 2** pages.
8.	The **Compare Pages** window's Page View pane. Brown highlights where checksum values differ in the **File 2**page from checksum values in the **File 1** page. Grayed-out items have matching checksum values in both the **File 1** and **File 2** pages.

Support Tools

About the Selection of Comparison Pages

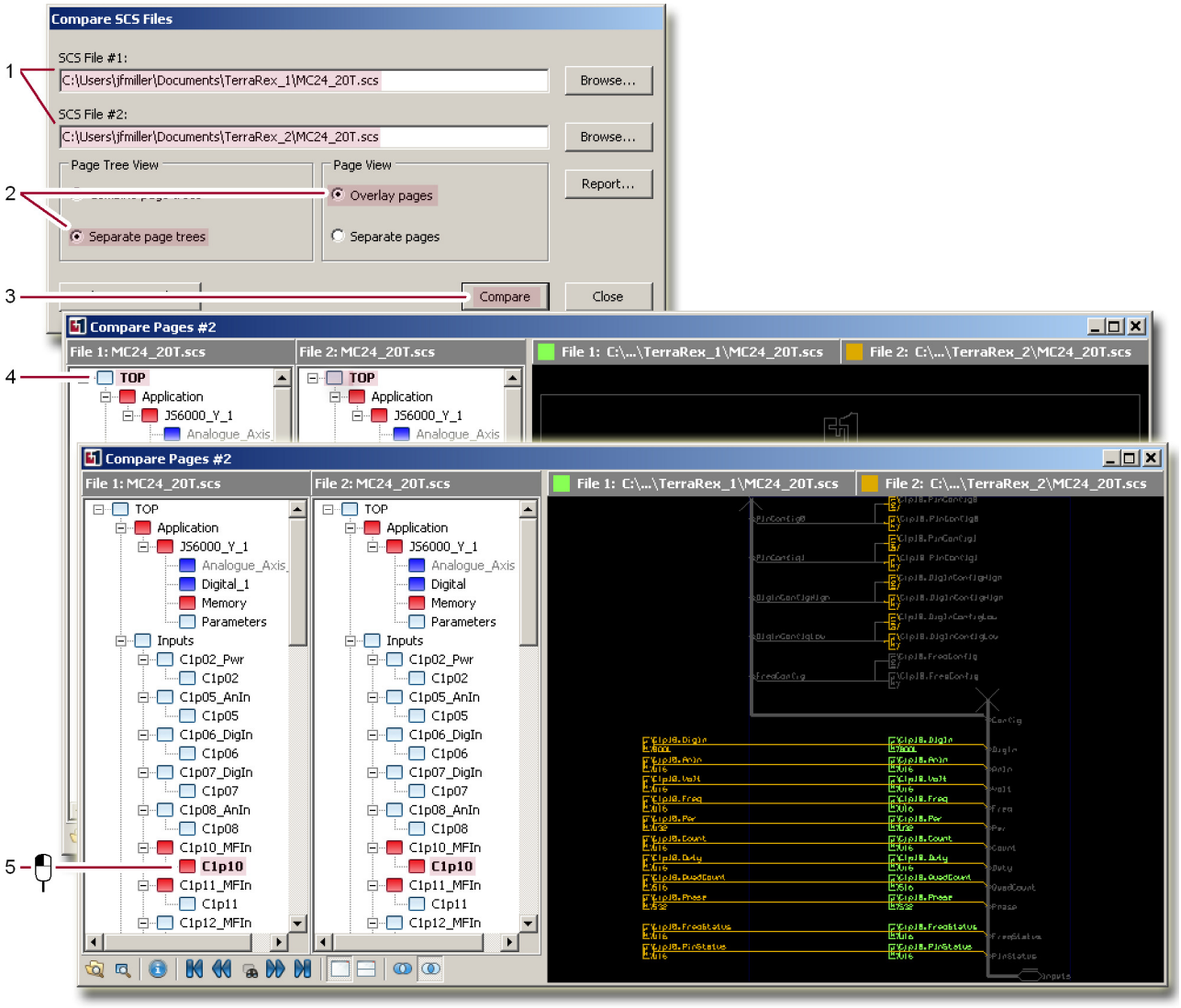


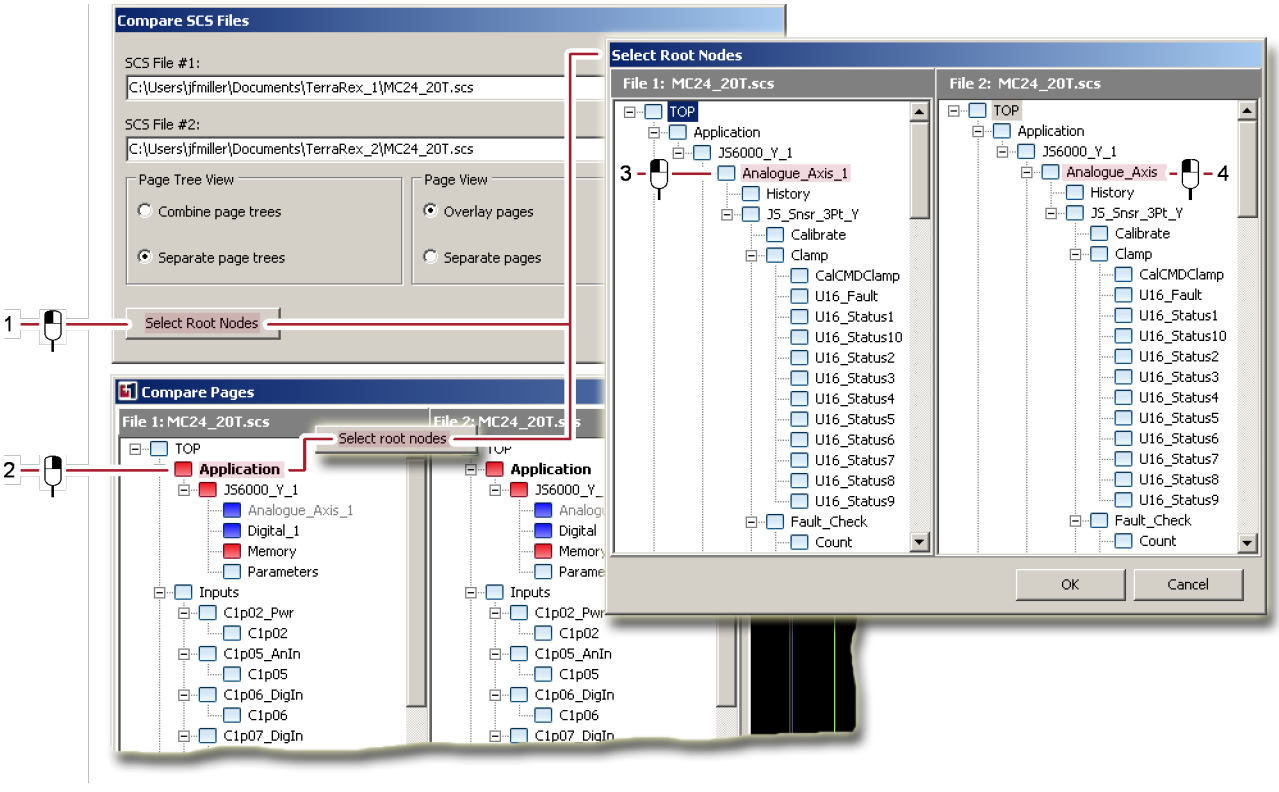
Figure Details

Item	Description
1.	Paths to the SCS files with the pages that you want to compare.
2.	Options that configure how the **Compare Pages** window shows page tree views and page views.
3.	The **Compare** button opens the **Compare Pages** window, with the **TOP** pages automatically selected as the default comparison pages.
4.	The TOP pages are the default comparison pages.
5.	Click to select another comparison page. The compare pages feature automatically selects the best comparison the other page tree view.

Support Tools

About the Select Root Nodes Window

Use the **Select Root Nodes** window to set the starting pages ("root nodes") from which a comparison starts. The PLUS+1® GUIDE software saves your root node settings.

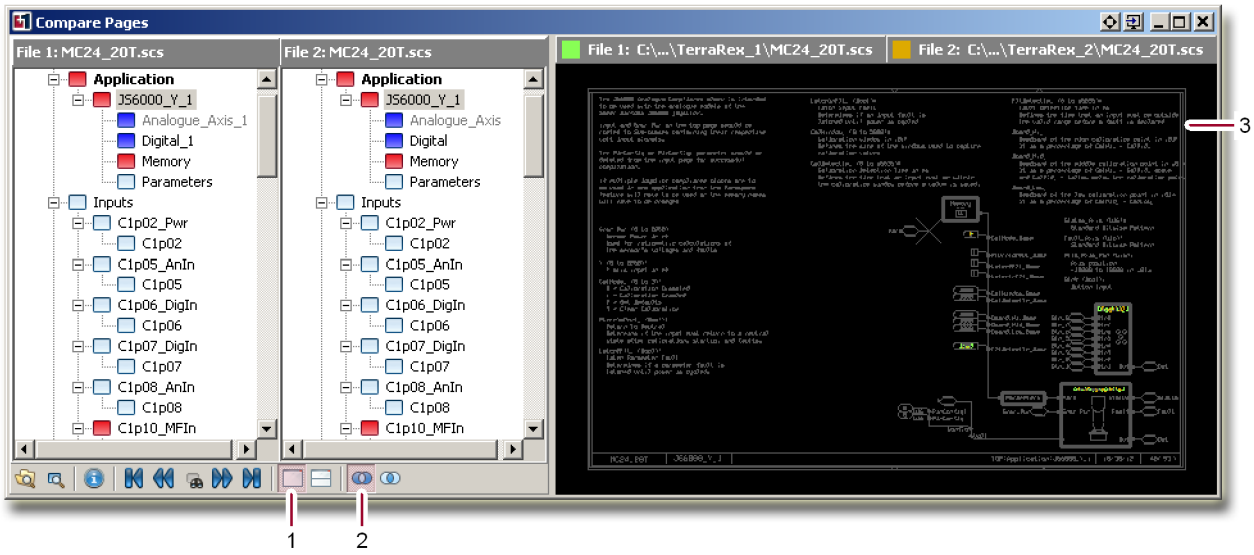


Select Root Nodes window description

Item	Description
1.	The **Select Root Nodes** button in the **Compare SCS Files** window opens the **Select Root Nodes** window.
2.	Right-clicking a page in the **Compare Pages** window displays a **Select root nodes** pop-up. The **Select root nodes** pop-up opens the **Select Root Nodes** window.
3.	In the **Select Root Nodes** window, clicking a page in the **File 1** tree view sets this page as the starting comparison for **File 1**.
4.	In the **Select Root Nodes** window, clicking a page in the **File 2** tree view sets this page as the starting comparison for **File 2**.
5.	The **OK** button closes the **Select Root Nodes** window. If you opened the **Select Root Nodes** window through the: • **Compare SCS Files** window's **Select Root Nodes** button, you return to the **Compare SCS Files** window. Use the **Compare SCS Files** window's **Compare** button to start a new comparison from the root nodes you have set. • **Compare Pages** window's **Select root nodes** pop-up, the **Compare Pages** window opens and shows a new comparison from the root nodes that you have set.

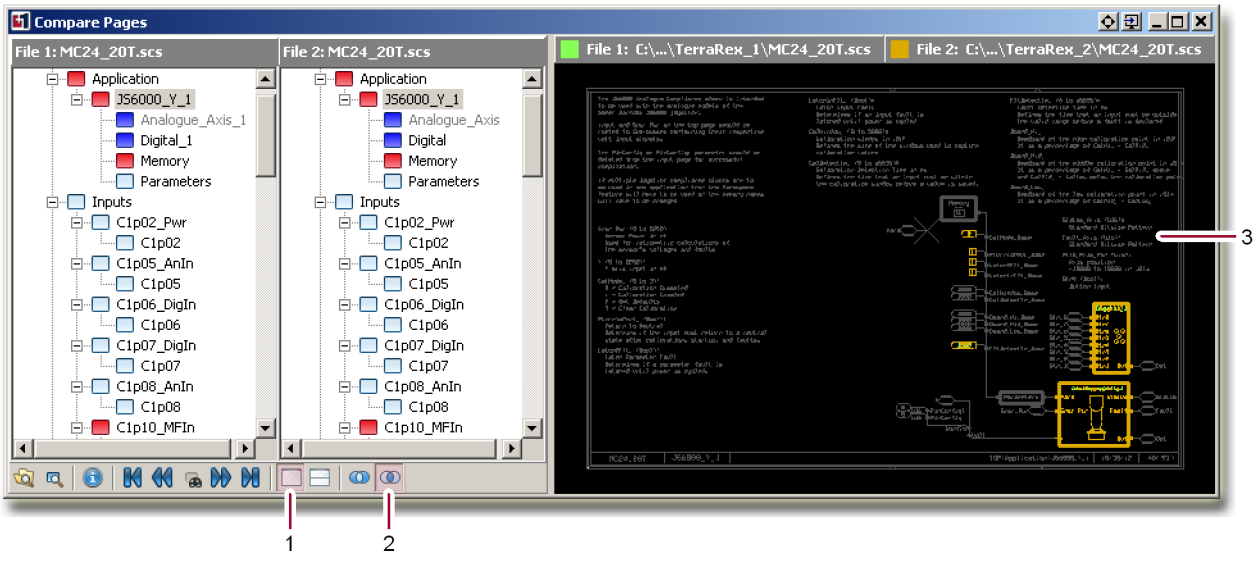
Support Tools

About Viewing Page – Example 1



1. The **Overlay Page View** button:
   - Shows overlaid page views of the selected comparison pages.
   - Enables the **Pixel Comparison View** and **Checksum Comparison** buttons.
2. The **Pixel Comparison View** button shows an XOR-like comparison of pixel positions in the **Page View** tab.
3. The Page View tab shows an XOR-like comparison of pixel positions. If a pixel position in the:
   - **File 1** page matches a pixel position in the **File 2** page, then both pixels gray out.
   - **File 1** page does not match a pixel position in the **File 2** page, then the unmatched pixel turns green.
   - **File 2** page does not match a pixel position in the **File 1** page, then the unmatched pixel turns brown.

About Viewing Page – Example 2



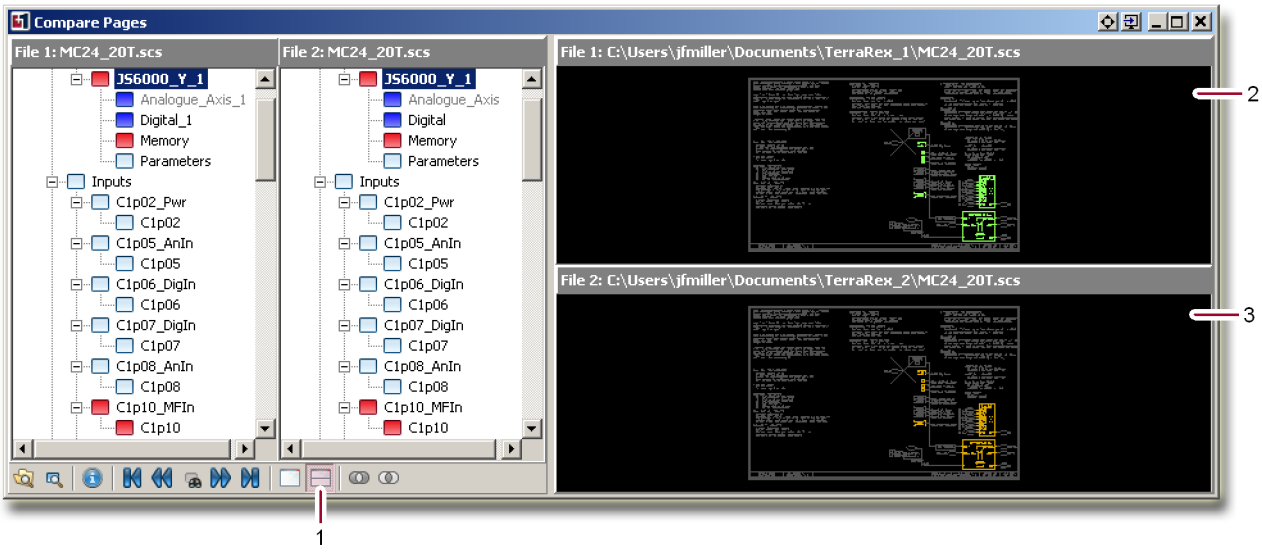
1. The **Overlay Page View** button:

Support Tools

- Shows overlaid page views of the selected comparison pages.
- Enables the **Pixel Comparison View** and **Checksum Comparison** buttons.

2. The **Checksum** button shows an XOR-like comparison of checksum values in the **Page View** tab.
3. The **Page View** tab shows an XOR-like comparison of checksum values. If a checksum for an item in the:
   - **File 1** page matches the checksum for an item in the **File 2** page, then both items gray out.
   - **File 1** page does not match the checksum for an item in the **File 2** page, then the unmatched item turns green.
   - **File 2** page does not match the checksum for an item in the **File 1** page, then the unmatched item turns brown.

About Viewing Page – Example 3



1. The **Separate Page View** button:
   - Shows separate page views of the selected comparison pages.
   - Disables both the **Pixel Comparison View** and **Checksum Comparison** buttons.
2. The upper Page View tab:
   - Highlights in green where checksum values differ in the **File 1** page from checksum values in the **File 2** page.
   - Grays out items that have matching checksum values in both the **File 1** and **File 2** pages.
3. The lower Page View tab:
   - Highlights in brown where checksum values differ in the **File 1** page from checksum values in the **File 2** page.
   - Grays out items that have matching checksum values in both the **File 1** and **File 2** pages.

Command Line Mode

Use the PLUS+1® GUIDE command line interface to perform tasks such as building projects, compiling changed projects, installing hardware files, exporting source code files, generating and running test code, as well as other tasks that can be helpful to build code on a continuous integration (CI) build agent.

Plus1_IDE

The executable is named `Plus1_IDE.exe` and it can be found in the **/P1Tools** installation folder.

Plus1_IDE can be run in 3 modes; GUI, CLI and GUI+CLI.

Support Tools

EXE modes

Modes	Description
GUI	PLUS+1® GUIDE will provide a standard Windows desktop user interface.
CLI	PLUS+1® GUIDE will provide a command line interface, without graphical elements.
GUI+CLI	PLUS+1® GUIDE will provide a combined interface where information will be provided both in the graphical and the command line interfaces.

Data Parameters; Command Modifier Parameters; and Command Parameters

Unless otherwise specified, parameters can be provided in any order.

Data Parameters

Name	Description
<project_file>	This marks a project file (*.p1x or *.p1p) that the command will target. Both full and relative paths are supported. A command can take at most 1 project file. When a project file is the only parameter, the project will simply be opened by PLUS+1® GUIDE in GUI mode.
<hwd_file>	This marks an HWD file that the command will target.
<hwd_partnumber>	This marks an HWD part number (including version) that the command will target.
<ccp_file>	This marks a CCP file that the command will target.
<sys_socket_description>	This marks a SYSSocket Description that the command will target.
<ekf_file>	This marks an EKF file that the command will target.
<csv_file>	This marks a CSV file that the command will target.
<rop_type>	This marks an ROP type that the command will target. The associated project must include an EXR file that has this type.
<zipped_file>	This marks a zipped file that the command will target. Supported file formats are .zip and .7z.
<SCS_file>	This marks an SCS file that the command will target.
<PLC_file>	This marks a PLC file that the command will target. Supported file formats are ***.xml**, ***.exp** and ***.st**.
<C_file>	This marks a C code file that the command will target. Supported file formats are ***.h** and ***.c.**
<LHX_file>	This marks a downloadable file generated by PLUS+1 GUIDE.
<sa_settings>	Static analysis settings file, in **.xml** format. Default value is the settings file used in GUIDE.

Command Modifier Parameters

Name	Description
-headless	Run PLUS+1® GUIDE in CLI mode only. Can only be used in combination with a command.
-keep-cli	Run PLUS+1® GUIDE in combined GUI+CLI mode. May be used in combination with a command, or just a project file.
-silent	Run PLUS+1® GUIDE in silent mode. In silent mode, PLUS+1® GUIDE will not output to stdout. May be used in combination with a command, or -keep-cli.

Support Tools

Command Modifier Parameters (continued)

Name	Description
-very-silent	Run PLUS+1® GUIDE in very silent mode. In very silent mode, PLUS+1® GUIDE will neither output to stdout nor to stderr. Should normally only be used in combination with -headless.
-generate-rop-exr	This command modifier must be combined with the -generate-rop-lhx command. It indicates that a read only parameter EXR file shall also be generated.
-verify-rop-exr	This command modifier must be combined with the -generate-rop-lhx command. It requires exactly 1 *rop_type* and 1 *project_file*. It indicates that a Read Only parameter `EXR` file shall also be verified.
-proceed-on-warnings	If the current command results in a warning which would normally require the user to confirm if the operation should proceed, then the command will instead proceed automatically. Can only be used in combination with a command.
-use-template	Can only be used with command -create-project. Instructs the command to use the HWD template as the main module of the created project.
-add-project-ekf	Can only be used with command -create-project. Can be used multiple times with command -create-project. Must be followed immediately by a <*sys_socket_description*> and then an <*ekf_file*>. Instructs the command to add an EKF to the project.
-sa-settings	Can only be used with the commands -static-analysis, -compile/-compile-changed, -compile-all/-build, -compile-sil-2, -error-check-all, -error-check-changed. Must be followed immediately by an <sa_settings>. Instructs the command to run static analysis with a specified settings file.
-overwrite-existing-file	Can only be used in combination with command -export-test-results. Instructs the command to overwrite the output file <*excel_file*> if it already exists.

Command Parameters

Name	Description
/?	Prints information about this command. When this parameter is provided, it must be the only parameter provided.
-compile-changed	Compile the specified <*project_file*> and then terminate. This command requires exactly 1 <*project_file*>. If compile succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1). Alias: -compile
-compile-all	Build (compile all) the specified <*project_file*> and then terminate. This command requires exactly 1 <*project_file*>. If build succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1). Alias: -build

Support Tools

Command Parameters (continued)

Name	Description
-compile-sil2	Build (compile all) the specified <*project_file*> for SIL2 and then terminate. This command requires exactly 1 <*project_file*>. This command can only succeed if the PLUS+1® GUIDE tool and the used HWD are both released for SIL2. Only use this command if you can attest that the application has been developed in accordance with SIL2 requirements. If build succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-install-hwd	Install the specified <*hwd_file*> and then terminate. This command requires exactly 1 <*hwd_file*>. If install succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-change-project-hwd	Change the HWD used in the specified <*project_file*> to the HWD specified by <*hwd_partnumber*> and then terminate. This command requires exactly 1 <*hwd_partnumber*> and 1 <*project_file*>. The HWD specified by <*hwd_partnumber*> must be already installed in PLUS +1® GUIDE. If the HWD change succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-change-project-ccp	Change the CCP used in the specified <*project_file*> to the CCP specified by <*ccp_file*> and then terminate. This command requires exactly 1 <ccp_file> and 1 <*project_file*>. If the CCP change succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-change-project-ekf	Change the EKF used in the specified <project_file> to the EKF specified by <sys_socket_description> and then terminate. This command requires exactly 1 <*sys_socket_description*>, 1 <*ekf_file*> and 1 <*project_file*>. If the HWD change succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-generate-rop-lhx	Use the specified <*csv_file*> to generate an output read only parameter LHX file. This command requires exactly 1 <*csv_file*>. If generation succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1). Note: this command can be combined with either -generate-rop-exr, or -verify-rop-exr, but not both at the same time.
-interactive	Run PLUS+1® GUIDE in interactive mode. May be used by itself, in combination with command modifiers, and/or with a project file. See description of interactive mode commands in the following table.
-pack-to-p1p	Pack the specified <*project_file*> into a packed P1P file, and then terminate. The project source files and temporary files will be deleted on disk. This command requires exactly 1 <*project_file*> in .p1x format. If packing succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-export-to-file	Export the source code files of the specified <*project_file*> into a zipped file and then terminate. This command will not modify any of the project files on disk. This command requires exactly one <*project_file*> and 1 <*zipped_file*>. If packing succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).

Support Tools

Command Parameters (continued)

Name	Description
-generate-test-code	Generate and compile test code for the specified <*project_file*> and then terminate. Only pages with tests that are not up to date will trigger generation and compilation of test code. This command requires exactly 1 <*project_file*>. If the command succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-run-tests	Execute test code for the specified <*project_file*> and then terminate. Only pages with test code that is up to date will trigger tests to be run. This command requires exactly 1 <*project_file*>. If the command succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-create-project	Used to create a new PLUS+1® GUIDE project from scratch. A P1X file and a <*hwd_partnumber*> are required. Additionally, any number of SCS files, PLC files and C/H files can be provided. The following modifiers are supported: -use-template, -headless, -keep-cli, -silent, -very-silent, -add-project-ekf If project creation succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-error-check-changed	Error check the specified <*project_file*> and then terminate. Only files that have changed, or depend on changed files, will be error checked. This command requires exactly 1 <*project_file*>. If Error check succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-error-check-all	Error check the specified <*project_file*> and then terminate. All files will be Error checked. This command requires exactly 1 <*project_file*>. If Error check succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-print-file-info	Prints information about a file. This command requires exactly 1 file parameter. No other parameters may be provided to this command. If print file info succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-apply-settings-file	Applies a GUIDE settings file to the current GUIDE installation. This command requires exactly 1 <zipped_file>. If command succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).
-static-analysis	Run static analysis on the specified <project_file> and then terminate. This command requires exactly one <project_file>. If command succeeds, then the exit code will be zero (0). Otherwise, the exit code will be one (1).

Interactive Command Parameters

Name	Description
open <project_file>	Opens a project in interactive mode
close	Closes the currently open project
compile	Compiles the currently open project
build	Builds (compiles all) the currently open project

Support Tools

Valid command combinations

Items within square brackets [ ] are optional.

The pipe character '|' means 'or'.

```
Plus1_IDE /?
Plus1_IDE <project_file>
Plus1_IDE -compile-changed [-headless | -keep-cli] [-proceed-on-warnings] [(-sa-settings <sa_settings>)] <project_file>
Plus1_IDE -compile-all [-headless | -keep-cli] [-proceed-on-warnings] [(-sa-settings <sa_settings>)] <project_file>
Plus1_IDE -compile-sil2 [-headless | -keep-cli] [-proceed-on-warnings] [(-sa-settings <sa_settings>)] <project_file>
Plus1_IDE -keep-cli [<project_file>]
Plus1_IDE -install-hwd <hwd_file> [-proceed-on-warnings]
Plus1_IDE -change-project-hwd <hwd_partnumber> <project_file> [-proceed-on-warnings]
Plus1_IDE -generate-rop-lhx <csv_file> [-generate-rop-exr | -verify-rop-exr <rop_type> <project_file>]
Plus1_IDE -change-project-ekf <ekf_file> <sys_socket_description> <project_file>
Plus1_IDE -change-project-ccp <ccp_file> <project_file>
Plus1_IDE -pack-to-p1p [-headless | -keep-cli] <project_file>
Plus1_IDE -export-to-file [-headless | -keep-cli] <project_file> <zipped_file>
Plus1_IDE -generate-test-code [-headless | -keep-cli] [-proceed-on-warnings] <project_file>
Plus1_IDE -run-tests [-headless | -keep-cli] <project_file>
Plus1_IDE -create-project <hwd_partnumber> <p1x_file> [-use-template] [-headless | -keep-cli] [-silent | -very-silent] [<SCS_file>+] [<PLC_file>+]
[<C_file>+] [(-add-project-ekf <sys_socket_description> <ekf_file>)+]
Plus1_IDE -error-check-changed [-headless | -keep-cli] [-silent | -very-silent] [(-sa-settings <sa_settings>)] <project_file>
Plus1_IDE -error-check-all [-headless | -keep-cli] [-silent | -very-silent] [(-sa-settings <sa_settings>)] <project_file>
Plus1_IDE -print-file-info (<LHX_file> | <project_file>)
Plus1_IDE -static-analysis [-headless | -keep-cli] [-silent | -very-silent] [(-sa-settings <sa_settings>)] <project_file>
```

ENGINEERING
TOMORROW

Products we offer:

- Cartridge valves
- DCV directional control valves
- Electric converters
- Electric machines
- Electric motors
- Gear motors
- Gear pumps
- Hydraulic integrated circuits (HICs)
- Hydrostatic motors
- Hydrostatic pumps
- Orbital motors
- PLUS+1® controllers
- PLUS+1® displays
- PLUS+1® joysticks and pedals
- PLUS+1® operator interfaces
- PLUS+1® sensors
- PLUS+1® software
- PLUS+1® software services, support and training
- Position controls and sensors
- PVG proportional valves
- Steering components and systems
- Telematics

Hydro-Gear
www.hydro-gear.com

Daikin-Sauer-Danfoss
www.daikin-sauer-danfoss.com

Danfoss Power Solutions is a global manufacturer and supplier of high-quality hydraulic and electric components. We specialize in providing state-of-the-art technology and solutions that excel in the harsh operating conditions of the mobile off-highway market as well as the marine sector. Building on our extensive applications expertise, we work closely with you to ensure exceptional performance for a broad range of applications. We help you and other customers around the world speed up system development, reduce costs and bring vehicles and vessels to market faster.

Danfoss Power Solutions – your strongest partner in mobile hydraulics and mobile electrification.

Go to www.danfoss.com for further product information.

We offer you expert worldwide support for ensuring the best possible solutions for outstanding performance. And with an extensive network of Global Service Partners, we also provide you with comprehensive global service for all of our components.

Local address:

Danfoss
Power Solutions (US) Company
2800 East 13th Street
Ames, IA 50010, USA
Phone: +1 515 239 6000

Danfoss
Power Solutions GmbH & Co. OHG
Krokamp 35
D-24539 Neumünster, Germany
Phone: +49 4321 871 0

Danfoss
Power Solutions ApS
Nordborgvej 81
DK-6430 Nordborg, Denmark
Phone: +45 7488 2222

Danfoss
Power Solutions Trading (Shanghai) Co., Ltd.
Building #22, No. 1000 Jin Hai Rd
Jin Qiao, Pudong New District
Shanghai, China 201206
Phone: +86 21 2080 6201

Index

C

D

F

I

L

M

Index