Wonderware® FactorySuite™

InControl™ User’s Guide

Revision H

Last Revision: January 2004

Invensys Systems, Inc.

Invensys Systems, Inc.
26561 Rancho Parkway South
Lake Forest, CA 92630 U.S.A.
(949) 727-3200
http://www.wonderware.com

Contents

InControl™ Language Editors User's Guide

1

Wonderware®

InControl™ Environment User's Guide

Revision H

Last Revision: August 2004

Invensys Systems, Inc.

Invensys Systems, Inc.
26561 Rancho Parkway South
Lake Forest, CA 92630 U.S.A.
(949) 727-3200
http://www.wonderware.com

Contents

CHAPTER 8: Running a Project....................143

C H A P T E R 1

Getting Started with InControl

This chapter contains the software and hardware requirements for InControl, instructions for installing and running InControl, how to get started programming, and Technical Support information.

Contents

Overview

InControl™ is one of the family of tools comprising the Wonderware® FactorySuite™ of process control solutions for factory floor applications. InControl is an open-architecture package that allows you to design, create, test, and run application programs for controlling your process. InControl is designed for close integration with the other FactorySuite components, including InTouch®, InBatch, and IndustrialSQL Server.

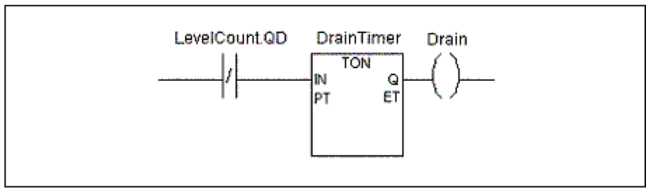
This manual has been updated for InControl Release 7.11.

IEC Compliance

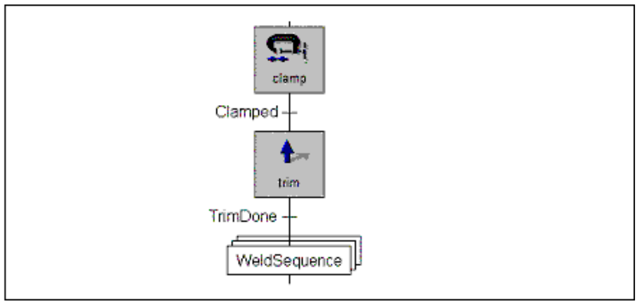
InControl incorporates the latest in international standards for designing your automation solution. InControl is compatible with the IEC-61131-3 international programming language specification. For more information, see the "Extensions to IEC 61131-3" appendix.

Programming Languages

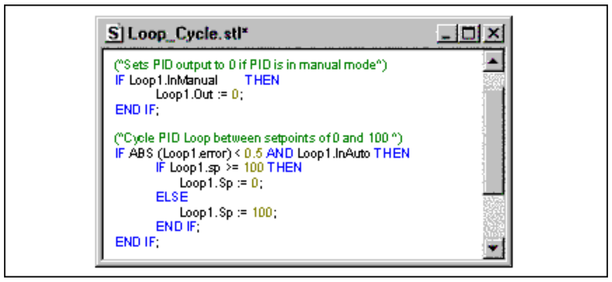
Create your factory automation solution in the following graphic- and text-based languages:



Relay Ladder Logic (RLL)



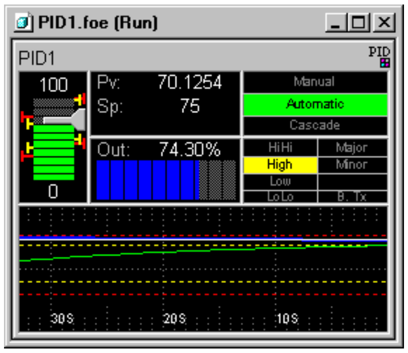
Sequential Function Chart (SFC)



```
(*Sets PID output to 0 if PID is in manual mode*)
IF Loop1.InManual      THEN
        Loop1.Out := 0;
END IF;

(*Cycle PID Loop between setpoints of 0 and 100 *)
IF ABS (Loop1.error) < 0.5 AND Loop1.InAuto THEN
        IF Loop1.sp >= 100 THEN
             Loop1.Sp := 0;
        ELSE
             Loop1.Sp := 100;
        END IF;
END IF;
```

Structured Text Language (STL)



Factory Object (FOE)

InControl is compatible with the ActiveX Server specification. The InControl Factory Object editor is an ActiveX container, which enables you to add ActiveX controls to a project. The current release of InControl includes several factory objects (FOEs). The following FOEs are described in this manual:

- Use the PID InControl FOE to handle PID loop functions.
- Use the Analog Alarm InControl FOE to monitor an analog input signal for alarm conditions.

These factory objects are described in the *InControl PID and Analog Alarm Reference Manual*.

For information about additional Wonderware Factory Objects, contact your distributor.

System Requirements

The InControl software is designed to run on any IBM compatible PC. Before installing InControl, verify that your system meets the following requirements.

InControl Project Development

- Any IBM® compatible PC with a Pentium II processor or higher (minimum: 400MHz on a single node system, recommended: 1.2GHz or higher).
- At least 2GB of free hard disk space.
- At least 256MB of random-access memory (RAM), 512MB of RAM is recommended.
- SVGA display adapter (2MB RAM recommended).
- Pointing device. For example, mouse, trackball.
- The Microsoft® Windows® 2000 Professional operating system with Service Pack 3 or higher or the Microsoft® Windows® XP Professional operating system with Service Pack 1 or higher or Microsoft® Windows® 2003 Enterprise server.

Wonderware InControl Version 7.11 SP2 (or later) does not support the Microsoft Windows 3.x, the Microsoft Windows for Workgroups, the Microsoft Windows 9x or the Microsoft Windows NT operating systems.

NetDDE is not supported with InControl on Windows Server 2003 (or later) operating systems.

Technical Support

Wonderware Technical support offers a variety of support options to answer any questions about Wonderware products and their implementation. Refer to the relevant chapters in your InControl documentation for a possible solution to any problem that you may encounter. If you find it necessary to contact InControl Technical Support for assistance, please have the following information available:

Before Contacting Wonderware

Refer to the relevant chapters in your InControl documentation for a possible solution to any problem that you may encounter. If you find it necessary to contact InControl Technical Support for assistance, please have the following information available:

- Software serial number and version number. Click **About InControl** on the **Help** menu, or **Configure** on the **Runtime** menu to determine the InControl version. You can also click **About Engine Monitor** on the runtime engine monitor icon.

  If InControl is not running, you can use the Windows Explorer to check the version properties (right-click the file name) of the file ICDev.exe or Rtengine.exe, located in the directory where InControl is installed.

  If InControl is not installed, you can determine the version by checking the file Version.txt, located in the InControl sub-directory of the distribution compact disc.

  The version number of an I/O driver appears in the title bar of the first configuration dialog box that appears during configuration.
- Support Contract Number. Your name must be one of the three contact names specified in the contract.
- Exact wording of system error messages encountered.
- Nature of problem and details of attempts made to solve the problem and results.
- System configuration information, e.g., operating system, and processor, RAM, hard disk size, etc.

Installation Guidelines

For a complete description of the InControl installation procedure, see the *InControl Installation Guide*.

InControl is distributed on a compact disc as a part of the FactorySuite. When you execute the Setup program, the system does the following:

- Creates the necessary directories on your hard disk, and copies the files into the appropriate directories.
- Sets up the program folder and icons.
- Modifies the system registry.

Before You Start—InTouch and InControl

If you intend to use InTouch with InControl on the same system, it is recommended that you install InTouch before installing InControl. If you install InControl first, you must run the InControl setup program again after installing InTouch in order to install the InControl extensions. To do this, rerun the InControl setup program. Click **Add/Remove** and then select **InTouch Extensions**.

If you intend to use InTouch and InControl on separate systems and need to view the InControl symbols from InTouch, install the files for the InTouch extensions on the system where InTouch is located. Run the InControl setup program on the InTouch system and select **InTouch Extensions** in the **Select Components** dialog box. It is not necessary to install any other components.

Before You Start—Additional Recommendations

Additional recommendations include the following:

- Before installing InControl, log on to your Windows system using an account with administrator privileges.
- Close any programs, particularly Wonderware applications, that are currently running.
- Uninstall any previous versions of InControl.

About the User Guides

The Setup program copies the InControl user guides and the individual I/O user guides to the Books subfolder under the FactorySuite folder.

Reading a Document

For information about how the user guides are installed, see the *FactorySuite System Administrator's Guide*. Be sure to check the Release Notes for additional information that does not appear in this user guide. This file is copied to your hard disk when you install InControl.

Contents - This User's Guide

For a quick introduction to the programming and runtime environments of InControl, see "Running InControl: Quickstart" of this chapter. Complete descriptions of the InControl environment (toolbars, menu options) are in the "InControl Environment" chapter. For information about configuring I/O, see the "I/O Configuration" chapter and the individual user's guide for each I/O driver. Detailed descriptions of the editors are given in the following chapters:

- RLL, STL, and SFC See the *InControl Language Editor User Guide*.
- InControl Factory Objects See the *InControl PID and Analog Alarm Reference Manual*.

Contents - I/O User's Guides

For specific information about the I/O drivers such as adding and configuring modules, devices, tags, etc., refer to the manuals for the individual drivers. The documentation for some third-party drivers appears only in the form of the online help that was developed by the manufacturer. For updates to this documentation, contact your distributor or the third-party manufacturer directly.

The HTML Help

This release of InControl includes online help in the form of a compiled HTML file. This new format provides several enhancements to the online help utility.

- You can modify the display format.
- A "Favorites" tab allows you to bookmark locations.
- A "Locate" button identifies the current open page within the table of contents.
- A "Zoom" button changes the size of the font for all text not in a table.

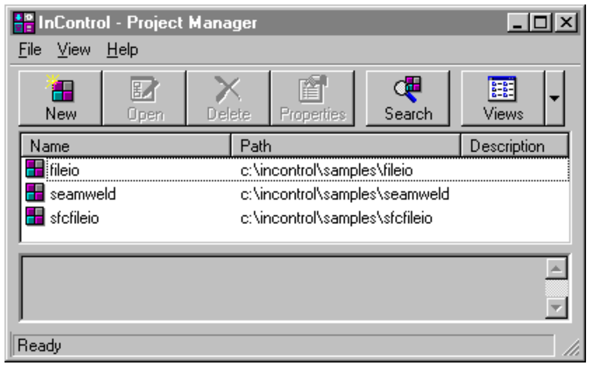
Note You can modify the display format by selecting display settings in the **Internet Options** dialog box. To open this dialog box, click **Options** on the Help toolbar and select **Internet Options**. Then click **Accessibility**. For the best results in displaying the online help, do not select format settings that ignore fonts and colors specified on Web pages. In addition, you may need to deselect the User Style Sheet checkbox.

Running InControl: Quickstart

Starting InControl

To run InControl:

1. Click **Start** on the Windows **Taskbar** to display the Start menu.
2. Point to **Programs\Wonderware FactorySuite**.
3. Click **InControl**. The **InControl Project Manager** dialog box appears:



Name	Path	Description
fileio	c:\incontrol\samples\fileio	
seamweld	c:\incontrol\samples\seamweld	
sfcfileio	c:\incontrol\samples\sfcfileio	

InControl Project Manager

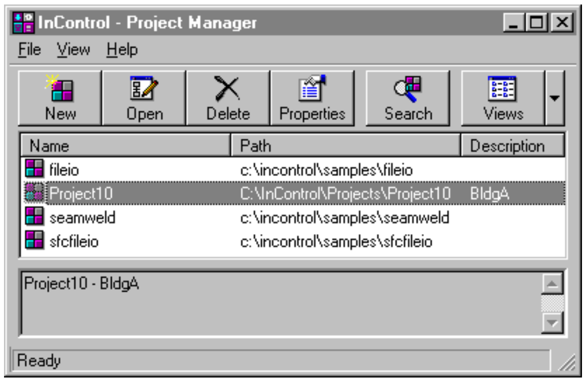
Creating a Project

InControl allows you to create groups of programs, called projects. Three projects appear in the figure above: Fileio, Seamweld, and Sfcfileio. All the programs within a project can be executed simultaneously, and you can coordinate them to handle your process. Before writing a program, you must create a project.

To create a project after starting InControl:

1. On the **File** menu of the **InControl Project Manager** dialog box, click **New**. The **Create InControl Project** dialog box appears.

2. Enter a project name, select a path, and click **OK**. In the following figure, Project10 has been created.


InControl - Project Manager
File View Help
New Open Delete Properties Search Views

Name	Path	Description
fileio	c:\incontrol\samples\fileio	
Project10	C:\InControl\Projects\Project10	BldgA
seamweld	c:\incontrol\samples\seamweld	
sfcfileio	c:\incontrol\samples\sfcfileio	

Project10 - BldgA

Ready

Project Manager and New Project

3. Double-click the project name to open the project in the Development environment. The **Runtime Engine Target** dialog box appears.
4. Select the target hardware platform and click **OK**. Unless you intend to run programs on another hardware platform, select the **Windows NT/ Windows 2000** target.

To create a Quickstart program, see the example program appendix in the language editor manual. These appendices describe how to write a simple RLL, SFC, and Structured Text program, function, and function block.

What's Next?

There is no predetermined method for doing the configuration and programming tasks for InControl. In general, the following order of tasks is recommended for most applications.

Order	Task	InControl Reference Manual Chapter
1	Add the I/O Drivers to the Project Window	"I/O Configuration"
2	Define I/O points.	Refer to the online I/O user's guides in the InControl Manuals directory
3	Define variables.	"Defining Variables"
4	Write application programs.	"Project Organization and Management" "InControl Language Editors"
5	Debug applications programs.	"Running a Project"
6	Run/Test application programs.	"Running a Project"
7	Configure security.	"Setting Up Security"
8	Design the HMI interface.	"InControl and InTouch" Also, refer to the *InTouch User's Guide.*

C H A P T E R 2

The InControl Environment

This chapter describes the InControl environment: toolbar items, menu options, screen fields, etc.

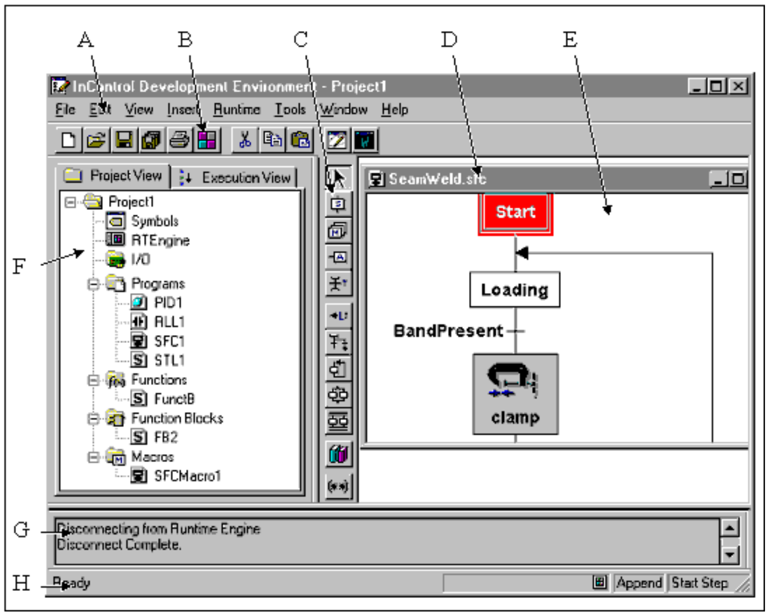
Contents

Working in the Development/Runtime Windows

InControl consists of two sets of windows: the development windows, where you create application programs, and the runtime windows, where you execute and monitor the programs that you create. You can control the various windows by hiding or showing them.

Development Window

A typical layout for the development windows is shown in the following figure.



Disconnecting from Runtime Engine
Disconnect Complete.

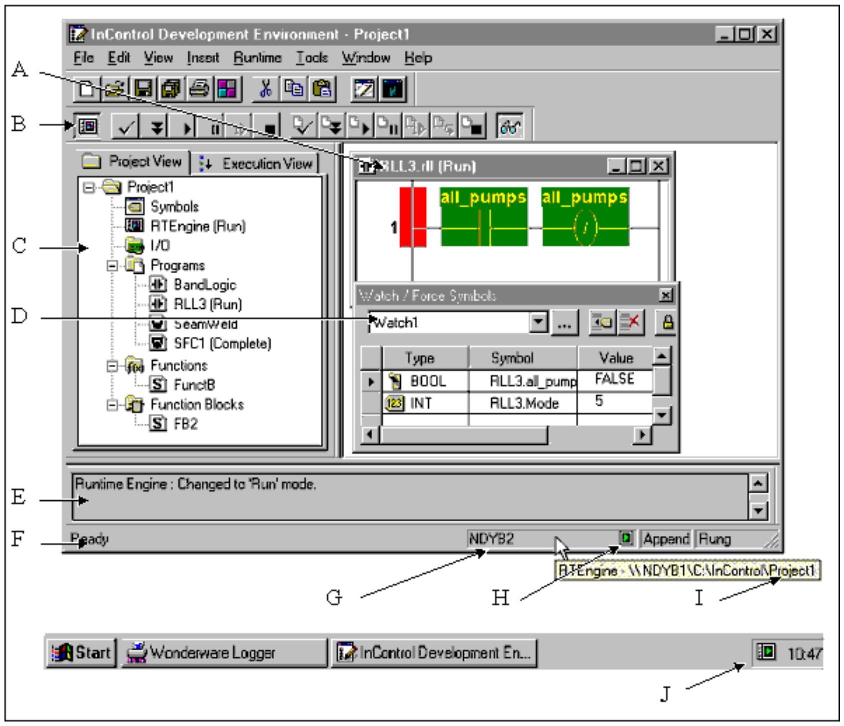
Development Window Screen Elements

Menu Bar (A)	Displays standard functions in a text format. Individual options are described in "Using the Menu Bar."
Standard Toolbar (B)	Displays standard functions as icons. Individual options are described in "Using the Standard Toolbar."
Editor Toolbar (C)	Displays the tools used to add program elements to a program. This toolbar changes, depending on the type of program being edited.
Editor Window Title bar (D)	Displays program name. An asterisk by a program name indicates that the program has been modified, but not saved, and/or is different from a copy running in the runtime engine. If a copy of the program is running, its mode (Run, Pause, Stopped) is also displayed. For functions, function blocks, and InControl factory objects that are downloaded to the runtime engine, the mode Loaded is displayed.
Editor Window (E)	Working area for the editors.

Project Window (F)	Displays project-specific functions: program and I/O organization and execution priority. Individual options are described in the"Project Organization and Management" chapter.
Output Window (G)	Displays messages, including error messages, from the runtime engine, the compiler, program elements, etc. This information is also written to the Wonderware Logger.
Status Bar (H)	Shows program information.

Runtime Window

A typical layout for the runtime window is shown in the following figure.



Runtime Window Screen Elements

Program Title Bar (A)	Displays program name. If a copy of the program is running, its mode (Run, Pause, Stopped, etc.) is also displayed. An asterisk by a program name indicates the following: **Not connected to the runtime environment**: The program has been modified, but not saved. **Connected to the runtime environment**: The program is different from a copy running in the runtime engine.
Runtime Toolbar (B)	Displays standard runtime functions as icons. Options are described in "Using the Runtime Toolbar."

Project Window (C)	Displays project-specific functions: program and I/O organization, I/O configuration, and execution priority. Individual options are described in the"Project Organization and Management" chapter.
Watch Window (D)	Displays variables (symbols) and their status at runtime. You can use the stand-alone Watch window if you do not want to open the Development/Runtime environment.
Output Window (E)	Displays messages from the runtime engine, the compiler, program elements, etc.
Status Bar (F)	Shows program information.
Connected Node (G)	Displays node to which the Development environment is connected.
Connected RTE Icon (H)	Displays the status of the runtime engine on the node, local or remote, that is running the project.
Downloaded Project (I)	Tool tip reports name of project downloaded to the runtime engine.
Runtime Engine Monitor Icon (J)	Monitors the status of the runtime engine on the local node. Can be used to control project mode and set scan times.

Runtime Engine Icons

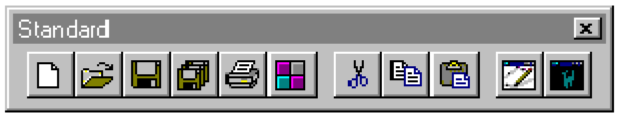
The following icons are associated with the runtime engine monitor icon and also appear in the Status Bar:

Runtime Engine Icons

Icon	**Function**
▶	Indicates the runtime engine is in Run mode (green).
⏸	Indicates the runtime engine is in Pause mode (yellow).
■	Indicates the runtime engine is in Stop mode (red).
🚫	Indicates the runtime engine is in the Fault mode. Note that this icon does not indicate a program is in the Fault mode.
❗	Indicates a message has been sent to the Output window and the Wonderware Logger. Icon also appears when a program enters the Fault mode or when the RLL MSGW or Structured Text MSGWND functions execute.
🔒	Indicates one or more variables are forced.

Using the Standard Toolbar

Several of the InControl functions can be selected from the standard toolbar, which is shown in the following figure.

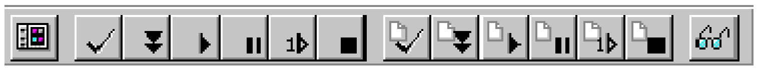


Standard Toolbar Options

Icon	Menu Bar Options	Function
🗋	**New** on the **File** menu.	Create a new program.
📂	**Open** on the **File** menu	Open an existing program.
💾	**Save** on the **File** menu.	Save the active program.
💾	**Save All** on the **File** menu.	Save all the open files.
🖨	**Print** on the **File** menu.	Print a program.
▦	**Project** on the **File** menu.	Open the Project Manager.
✂	**Cut** on the **Edit** menu.	Cut the selected object and place it on the clipboard.
📄	**Copy** on the **Edit** menu.	Copy the selected object and place it on the clipboard.
📋	**Paste** on the **Edit** menu.	Paste the contents of the clipboard.
✎	**WindowMaker** on the **Tools** menu.	Access the InTouch Window Maker.
■	**WindowViewer** on the **Tools** menu.	Access the InTouch WindowViewer.

Using the Runtime Toolbar

The InControl runtime commands can be selected from the Runtime Toolbar, which is shown in the following figure.



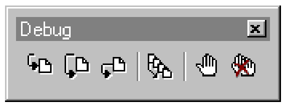
Runtime Toolbar Commands

Icon	Runtime Menu Command	Description
▦	Connect / Disconnect	*
✓	Validate Project	*
⏬	Download Project	*
▶	Run Project	*
⏸	Pause	*
1▷	Single Scan	*
■	Stop	*
✓	Validate Program	*
⏸	Download Program	*
▶	Run Program	*
⏸	Pause Program	*
1▷	Single Scan Program	*
■	Stop Program	*
👓	View Menu: Watch/Force Variables	Display Watch Window. You can use the stand-alone Watch window if you do not want to open the Development/ Runtime environment.
* For a detailed description of the command's function, see "Runtime Commands."		

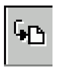
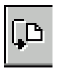
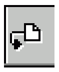
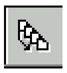
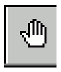
For information about running projects and programs, see the "Running a Project" chapter.

Using the Debug Toolbar

The InControl debug commands can be selected from the Debug Toolbar, which is shown in the following figure.



Debug Toolbar Commands

Icon	Debug Menu Commands	Description
	Step Into Program.	*
	Step Over Program.	*
	Step Out Program.	*
	View Call Stack.	*
	Toggle Breakpoint.	*
	Clear All Breakpoints.	*
* For a detailed description of the command's function, see "Debug Commands."		

For more information about debugging programs, see the "Running a Project" chapter.

Using the Menu Bar

You can choose any of the InControl functions from the menu bar, which is shown in the following figure.

File Commands

Use these commands for file operations, such as opening, closing, and printing files, and adding or removing a program file from a project.

File Menu Commands

File Menu Command	Toolbar Icon	Function
New	🗋	Create a new program file.
Open	📂	Open an existing program file.
Close	n/a	Close all the windows associated with the active program file.
Save	💾	Save the active program file.
Save As	n/a	Save the active program under a different name.
Save All	💾	Save all open program files and related project information.
Project	▦	Open the Project Manager.
Add File to Project	n/a	Add a program file to a project.
Remove File From Project	n/a	Remove a program file from a project.
Print	🖨	Print a program.
Print Xref [1] (Print Cross References)	n/a	Print program variables, where and how often they are used in the program. Available only for RLL and SFC programs.
Print Setup	n/a	Allows you to change the printer and printing options.
Exit	n/a	Close the Development environment. If unsaved programs are open, you are prompted to save them.

1 You can also print cross-references from the Symbol Manager. For more information, see the "Defining Variables" chapter.

Edit Commands

The Edit menu appears only when a program is open for editing. Commands change depending on the type of program in the currently active window. Program-specific commands are noted. InControl indicates that a command has been selected by placing a check by it.

Edit Menu Commands

Edit Menu Command	**Specific Program**	**Function**
Undo	All	Undo the last action.
Redo	RLL SFC STL	Redo the previously undone action that involves a program element.
Cut	All	Cut the selected object and place it on the clipboard.
Copy	All	Copy the selected object and place it on the clipboard.
Paste	All	Paste the contents of the clipboard.
Delete	All	Delete the selected object.
Select All	SFC-Stp STL	Select all the lines of code.
Mark Line	SFC-Stp STL	Use to select one or more lines of code.
Edit Element	RLL SFC	Open the dialog box for the selected program element.
Step Properties	SFC	Open the dialog box for a Step.
Find	All	Find the specified text.
Find Next	All	Find the next occurrence of the specified text.
Replace	All	Replace the specified text with new text.
Go To	SFC-Stp RLL STL	Display selected location in the program.
Go To Coil	RLL	Display the next occurrence of the selected coil.
Boolean Transition	SFC	Set the default Transition for all open SFC programs to the Boolean type (checked), instead of the RLL type (unchecked). This command does not change existing Transitions.
Lock Algorithms	SFC	Use to add password protection to Step algorithms. When an algorithm is locked, this command is "Unlock Algorithm."
Set Bookmark	SFC-Stp STL	Mark one or more locations in a program and use the Go To command to jump between them.
Complete Symbol	SFC-Stp STL	Opens Symbol Picker for fast entry of symbol names.
Properties	FOE	Display the configuration dialog boxes.

Edit Menu Command	Specific Program	Function
Events	FOE	Open the Events Editor for the FOE.
Upload Configuration	FOE	Upload parameters from the runtime engine to the Development environment.

RLL = Relay Ladder Logic
FOE = InControl Factory Object
SFC = Sequential Function Chart
STL = Structured Text
SFC-Stp = SFC Step		

View Commands

The View menu lists screen elements that you can hide or display. Elements change depending on the type of program in the currently active window. Program-specific elements are noted. InControl indicates that an element has been selected by placing a check by it.

View Menu Commands

View Menu Command	Specific Program	Function
Toolbar	All	Display program functions as icons. The standard toolbar is illustrated in "Development Window."
Runtime Toolbar	All	Display runtime functions as icons; toolbar appearsautomatically when connecting to the runtime engine. The runtime toolbar is illustrated in "Using the Runtime Toolbar."
Debug Toolbar	All	Display the debug functions as icons.
Status Bar	All	Display the current mode of the runtime engine (Running, Paused, Stopped), edit mode (insert or append), the current cursor location in the program currently being edited, and help information when the cursor is over a button. The status bar is illustrated in "Runtime Window."
Zoom Toolbar	All	Display the zoom functions as icons.
Contact/Coil Bar	RLL	Display the program elements as icons. The Contact/Coil bar is described in the"Using the RLL Editor" chapter.
SFC Bar	SFC	Display the program elements as icons. The SFC bar is described in the "Language Editors" chapter.
Structured Text Toolbar	SFC-Stp STL	Display the program elements as icons. The Structured Text bar is described in the "Language Editors" chapter.

View Menu Command	Specific Program	Function
Factory Object Bar	FOE	Display the configuration dialog boxes. The Factory Object bar is described in the "Using the Factory Object Editor" chapter.
Block Palette	RLL	Display the function blocks that can be used in the program.
Project	All	Display the working area for the functions in the Project window. The Project window functions are described in the "Project Organization and Management" chapter.
Output	All	Display messages issued by the runtime engine, the compiler, program elements, etc. The Output Window is illustrated in "Runtime Window."
Watch/Force Variables	All	Display selected variables and their current values at runtime. Use to specify new values for variables or force variables to help in debugging a program.
Logger	All	Display the Wonderware Logger, which keeps a record of runtime messages. These messages also appear in the Output window.
Program Comments	RLL SFC	Display or hide any program comments that you enter into the program. Available only when a program is open for editing. For RLL programs, you must choose this option before entering a new comment.
Symbol Addresses	RLL SFC	Display addresses for I/O points in RLL programs and in SFC Actions.
Function Block Details	RLL	Display RLL function block variables. When the program is running, the contents of the variables are updated.
Rung Wrapping	RLL	Wrap RLL networks so that they can be viewed in the editor window.
Auto Pagebreak	RLL	Force RLL network to fit on a page when printing. Also operates in the editor.
All Steps	SFC	Display Steps by showing their names, descriptions, associated icons, or their program code.
Runtime Highlighting	RLL SFC FOE	Enable the runtime animation for a program. Set the update frequency for the animation in the **Runtime Engine Properties** dialog box.

View Menu Command	Specific Program	Function
Zoom	RLL SFC	Zoom in or zoom out of the program.

RLL = Relay Ladder Logic
SFC = Sequential Function Chart
FOE = InControl Factory Object
STL = Structured Text
SFC-Stp = SFC Step		

Insert Commands

The Insert commands allow you to place program elements into a program from the Menu Bar, instead of from the program toolbars. For more information about the program elements, see the appropriate chapter for the program type.

Runtime Commands

The Runtime commands allow you to set runtime parameters, observe runtime status data, and to send commands directly to the runtime engine. See the "Running a Project" chapter for more information about how these commands are used.

Runtime Commands

Runtime Menu Command	Toolbar Icon	Description
Connect Disconnect	▦	Connect the Development environment to the runtime engine. The engine runs continually as a Windows service, and whether it actually executes a project as it runs, depends on its mode of operation (Run, Stop, Pause, etc.). When the runtime engine is connected, the icon is depressed and the option is “Disconnect,” which disconnects the Development environment from the runtime engine. If you close (exit) the Development environment, the runtime engine continues to run.
Configure	n/a	Display the **Offline Runtime Engine Properties** dialog box if not connected to the runtime engine. Display the **Online Runtime Engine Properties** dialog box if connected to the runtime engine.
Report Status	n/a	Examine runtime engine status data, such as current project, time stamp, scan time, mode, processor utilization, faulted programs, I/O faults, etc. This data appears in the Output window and the Wonderware Logger.

Runtime Menu Command	**Toolbar Icon**	**Description**
Clear Faults	n/a	Set faulted programs to Pause mode, clear I/O faults, and clear runtime engine error status bits, such as RTEngine.ScanOverrun.
Validate Project	✓	Validate all programs in a project. All modified programs are saved to the hard disk.
Download Project [1]	⏬	Download all programs in a project to the runtime engine. Modified programs are saved to the hard disk. Programs are validated if necessary.
Upload Project Values	n/a	For all programs, replace defined initial values (for all local and global variables) with current values in the runtime engine. Does not upload I/O variables, arrays, or values that you cannot define during configuration, e.g., the Mode symbol. The Output window displays data that is uploaded.
Run Project [1]	▶	Run all programs in a project. Programs are validated and downloaded if necessary. All modified programs are saved to the hard disk.
Pause	⏸	Pause all programs that are currently being run by the runtime engine. The I/O continues to be updated.
Single Scan	1▷	Execute a single scan of the runtime engine. I/O is updated, then all programs in a project that are currently downloaded to the runtime engine are executed one scan. Can only be done while runtime engine is paused. See the "Running a Project" chapter for more information.
Stop	■	Stop all programs in a project that are currently being run by the runtime engine. Programs are unloaded from memory. The I/O goes to the state defined in the configuration for each I/O board.
Validate Program	🗋✓	Validate selected program. If program was modified, it is saved to the hard disk.
Upload Program Values	n/a	For currently selected program, replace defined initial values of local variables with current values in the runtime engine. Does not upload arrays, or values that you cannot define during configuration, e.g., the Mode symbol. The Output window displays data that is uploaded.
Download Program	🗋⏬	Download selected program to the runtime engine. If program was modified, it is saved to the hard disk.
Run Program	🗋▶	Run the selected program. If program was modified, it is saved to the hard disk.

Runtime Menu Command	**Toolbar Icon**	**Description**
Pause Program	⏸	Pause a program that is currently being run by the runtime engine.The I/O continues to be updated.
Single Scan Program	1▷	Execute a single scan of the program. I/O is updated and then the selected program is executed one scan. Can only be done while the program is paused.
Stop Program	⏹	Stop a program and unload it from memory. Other programs in the project and I/O are unaffected.
1 Programs that have been excluded from the project load on their property sheets are not downloaded or run.		

Debug Commands

The Debug commands allow you to locate and correct errors in your code. See the "Running a Project" chapter for more information about how these commands are used.

Debug Commands

Debug Menu Commands	Toolbar Icon	Description
Step Into Program.		With program paused, click to execute a single line of code. If the code calls a function, execution takes place at the first line of the function.
Step Over Program.		With program paused: RLL: Execute one rung. Structured Text: Execute one line of code. SFC: Execute one line of code of every active Step. All active Actions are executed completely. If program flow has been paused at a function call, the call is skipped. Structured Text and SFC programs must be compiled with the **Debug Enabled** option selected.
Step Out Program.		With program paused, click to cause program flow to leave a function that has been called. Program flow resumes at the line following the function call.
View Call Stack.		Click to show current sequence of function calls.
Toggle Breakpoint.		With program paused, enables/disables breakpoint at the selected line of STL code.
Clear All Breakpoints.		With program paused, disables all breakpoints that have been enabled.

For more information about debugging programs, see the "Running a Project" chapter.

Tools Commands

Use the Tools commands to call other programs or utilities for execution. Tools Menu Commands

Tools Menu Command	Specific Program	Function
Symbol Manager	All	Access the Symbol Manager; use to create and edit variables (see the "Defining Variables" chapter).
Action Manager	SFC	Access the Action Manager; use to rename and delete SFC Actions (see the "Using the SFC Editor" chapter).
RLL Transition	SFC	Access the Transition Manager; use to rename and delete SFC RLL Transitions (see the "Using the SFC Editor" chapter).
Step Library	SFC	Access the Step Library; use to create, edit and delete predefined Steps in the library.
InTouch	All	Display the opening menu for InTouch.
WindowMaker	All	Access the InTouch WindowMaker.
WindowViewer	All	Access the InTouch WindowViewer.
Security	All	Access the Security Manager; use to configure system security (see the "Setting Up Security" chapter).
Configure Colors	SFC, STL, RLL	Access the color selection dialog box; use to specify colors for the programs in the editors and at runtime.
Configure RLL/SFC Font Configure Text Editor Font	RLL, SFC STL	Access the font selection dialog box; use to specify font used for the programs in the editors and at runtime.

Window Commands

Use the Window commands to organize windows. The Window menu does not appear unless you have opened an editor window.

Window Commands

Window Menu Command	Function
Cascade	Arrange windows so they overlap.
Tile	Arrange windows as non-overlapping tiles.
Arrange Icons	Move icons to the bottom of the active window.
Close All	Close all editor windows. Unsaved programs are saved.

Help Commands

Use the Help commands to display system information and help options.

Help Commands

Help Menu Command	Function
InControl Help Topics	Display the table of contents for system Help.
SFC Editor Help	Display Help for the SFC editor.
RLL Editor Help	Display Help for the RLL editor.
Structured Text Editor Help	Display Help for the Structured Text editor.
Wonderware via Internet	For systems with access to the Internet, connect to the Wonderware Web Page.
About InControl	Display your serial number and the current version of InControl.
About SFC Editor	When an SFC window is active, display current version of the SFC editor.
About RLL Editor	When an RLL window is active, display current version of the RLL editor.
About Structured Text Editor	When a Structured Text window is active, display current version of the Structured Text editor.
About Factory Object Editor	When an FOE window is active, display current version of the FOE editor.

C H A P T E R 3

Setting Up Security

This chapter describes the InControl environment: toolbar items, menu options, screen fields, etc.

Contents

Overview

Access to the InControl environment is restricted to help ensure that only authorized and/or qualified people can interact with the factory process. These three levels of security access are available: Administrator, Engineer, and Operator. The following table shows the security level required to do the listed tasks. A **Y** indicates the security level that is required to do the task.

Task	Security Level		
	Administrator	**Engineer**	**Operator**
Edit Program	Y	Y	
Edit I/O Configuration	Y	Y	
Edit Symbols	Y	Y	
Start/Stop Program	Y	Y	Y
Download Program	Y	Y	
Change Passwords	Y		
Add/Delete User Names	Y		
Modify/Force Values	Y	Y	

InControl is shipped with one default user name: Administrator. The default password is an empty string. When prompted, press **Enter**. The system administrator, with the Administrator security level, assigns all users and passwords for the system. It is highly recommended that the password be changed for the Administrator user after you have installed the InControl software.

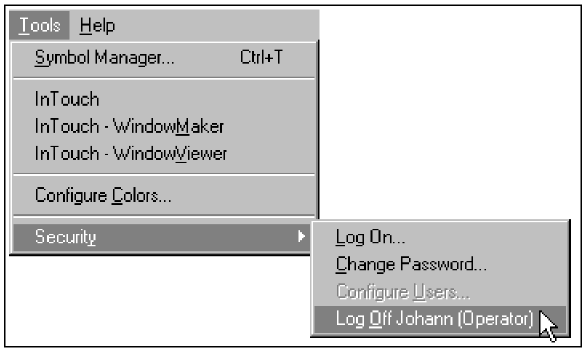
WARNING! Users who have unlimited access to all the InControl configuration tasks may be able to make inappropriate and unauthorized changes to the system. This could cause unpredictable operation by the controller, which can result in death or injury to personnel and/or damage to equipment.

Be sure to change the default password for the Administrator security level and advise all users to protect their assigned passwords.

InControl security is designed to work with the security features that are a part of the Windows Operating System. Your systems administrator can help ensure system integrity by configuring user Ids so that only authorized users have access to InControl projects and the runtime engine.

See the documentation for the Windows operating system for information about how to configure this level of security.

To check the security level for the user who is currently logged on, click **Security** on the **Tools** menu. The user and security level appear in the menu, as shown in the following figure.



Checking Security Level

Logging On/Off and Changing a Password

This section describes the security tasks that are typically done by any user, including an operator.

Logging On

To log on:

1. On the **Tools** menu, point to **Security** and click **Log On**. The **Log On** dialog box appears.
2. Enter your name and password. Click **OK**.

   If another user is logged on, that user is automatically logged off.

Logging Off

To log off:

1. On the **Tools** menu, point to **Security**.
2. Click **Log Off xxx**. If another user is logged on, this option displays the name of the current user in the *xxx* field.
3. If no users are logged on, the **Log Off** option is unavailable.

Changing a Password

A user can change the password used with a user name for logging on.

To change the password for a user name:

1. Log on with the user name for which you are changing the password.
2. On the **Tools** menu, point to **Security**, and click **Change Password**. The **Change Password** dialog box appears.
3. Enter the current password.
4. Enter the new password and then reenter it to verify.
5. Click **OK** to close the dialog box.

Managing Security

This section describes the tasks that are typically done by the system administrator.

Immediately after you install InControl and run the program, you have the Administrator's security access level. You can begin doing normal tasks, e.g., configuration, writing and editing programs, without logging on. The system remains in this state until you log off or make changes in security. After this, you must log on before you can make any changes.

Adding User Names

To make changes in the security configuration, you must have access to the Change Passwords security task.

To add a user name:

1. Log on as administrator.
2. On the **Tools** menu, point to **Security**, and click **Configure Users**. The **Configure Users** dialog box appears.
3. Enter the new user name in the **User Name** field.
4. Enter the password in the **Password** field.
5. Select the access level in the **Access** field (Administrator, Engineer, or Operator).
6. Click **Require logon at startup** to require users to log on when running InControl. Note that this option affects all users, as well as the one being added.
7. Click **Add**. The new user name is added to the security access level list.

Changing Passwords and Deleting User Names

To make changes in the security configuration, you must have access to the Change Passwords security task.

To change a user's password or security access level:

1. Log on as administrator.
2. On the **Tools** menu, point to **Security** and click **Configure Users**. The **Configure Users** dialog box appears.
3. Click the user name that you are changing.
4. If you are changing the password, enter the new password in the **Password** field.
5. If you are changing the security access level, select the new access level in the **Access** field.
6. Click **Update**. The new information is added to the security access level list.
7. Click **OK** to close the dialog box.

To delete a user name:

1. Log on as administrator.
2. On the **Tools** menu, point to **Security** and click **Configure Users**. The **Configure Users** dialog box appears.
3. Click the user name that you are deleting.
4. Click **Delete**. The user name is removed from the security access level list.
5. Click **OK** to close the dialog box.

Locking SFC Algorithms

InControl allows you to protect program code within an SFC Step from unauthorized changes. Select the **Lock Algorithms** command in the **Edit** menu and assign a password. To lock the SFC code, you must have access to the Edit Program security task.

To lock SFC algorithms:

1. Open the SFC program.
2. On the **Edit** menu, click **Lock Algorithms**. The **Enter Lock Password** dialog box appears.
3. Enter a password in the **Password** field. This field is case sensitive. Use a password that is different from the security password.
4. Confirm the password and click **OK**.
5. When prompted to save and lock the active file, click **Yes**.

Locking the algorithms prevents unauthorized users from changing the Structured Text code within the program's Steps. It does not prevent them from editing other elements in the program, e.g., Transitions, Actions, Loops, etc., and it does not prevent unauthorized users from deleting Steps.

WARNING! SFC Step code that has been locked cannot be changed without the password. If you forget the password, you cannot edit the code for any Step in the program. Be sure to keep a copy of the password in a safe place.

To unlock SFC algorithms:

1. Open the SFC program.
2. On the **Edit** menu, click **Unlock Algorithms**. The **Validate Unlock Password** dialog box appears.
3. Enter the password in the **Password** field and click **OK**.

   Unlocking the algorithms allows all users with access to the Edit Program security task to edit the Structured Text within the program's Steps.

Using Windows Security

You can restrict access to a hardware unit by requiring all operators and other users to log on to the Windows Operating System. Your Windows systems administrator can configure user Ids so that only authorized users can run selected programs and access selected directories. With the proper security configuration, the runtime engine monitor menu, which allows access to the runtime engine, is not displayed and cannot be accessed without the appropriate password.

See the documentation for the Windows operating system for information about how to configure this level of security.

C H A P T E R 4

I/O Configuration

This chapter describes the general approach to configuring I/O scanner boards. For more detailed information, refer to the individual I/O Configuration documents and the third-party user documentation that accompanies the I/O board.

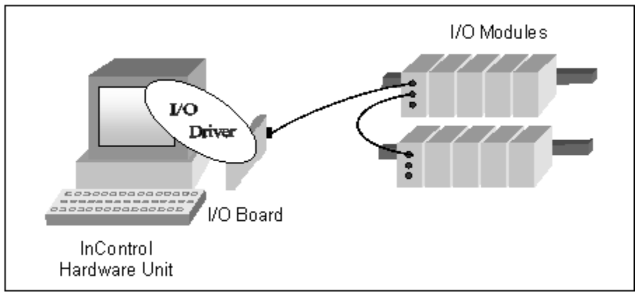
Contents

Overview

Typically, a physical board must be installed in your system for InControl to communicate with factory floor I/O devices. The following pages give examples of various types of scanner board installations.

Single Board Installation

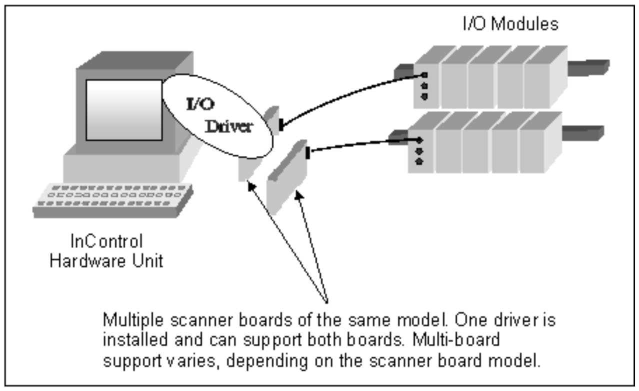
The following figure illustrates a third party I/O scanner board installed in the InControl hardware unit to scan the I/O modules.



Single Board Installation

Multiple Board Installation

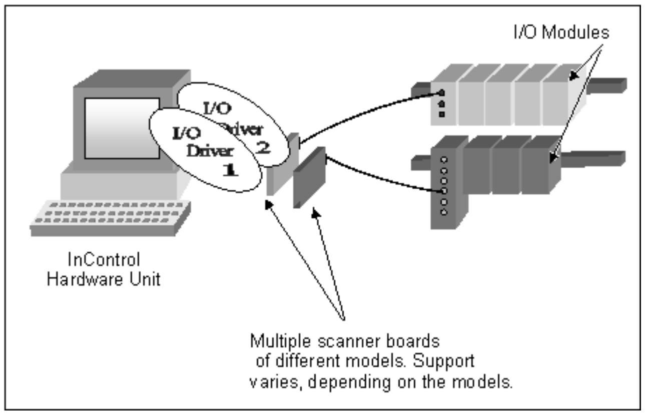
Depending on the scanner board model, you may be able to install multiple boards of the same model in the InControl hardware unit, as illustrated in the following figure.


I/O Modules
I/O Driver
InControl Hardware Unit
Multiple scanner boards of the same model. One driver is installed and can support both boards. Multi-board support varies, depending on the scanner board model.

Multiple Board Installation

Multiple Board / Different Vendor Installation

For some scanner board models, you may be able to install boards from different manufacturers in the InControl hardware unit, as illustrated in the following figure.

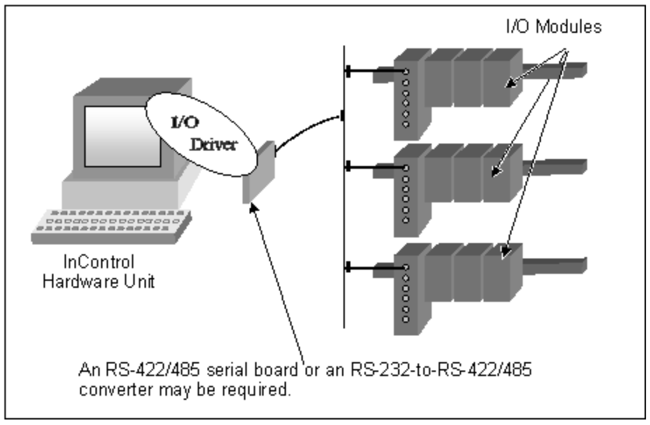

I/O Modules
I/O Driver 2
I/O Driver 1
InControl Hardware Unit
Multiple scanner boards of different models. Support varies, depending on the models.

Multiple Board/ Different Vendor Installation

Note Check with the manufacturers to determine whether their scanner boards can operate in the same hardware unit with other scanner boards.

Communicating Without a Scanner Board

Some I/O drivers do not require a specific scanner board to be installed in the InControl hardware unit. The following figure illustrates how an I/O driver communicates with an intelligent I/O module directly through a serial board.


I/O Modules
I/O Driver
InControl Hardware Unit
An RS-422/485 serial board or an RS-232-to-RS-422/485 converter may be required.

Communicating Without a Scanner Board

Communicating Through SuiteLink

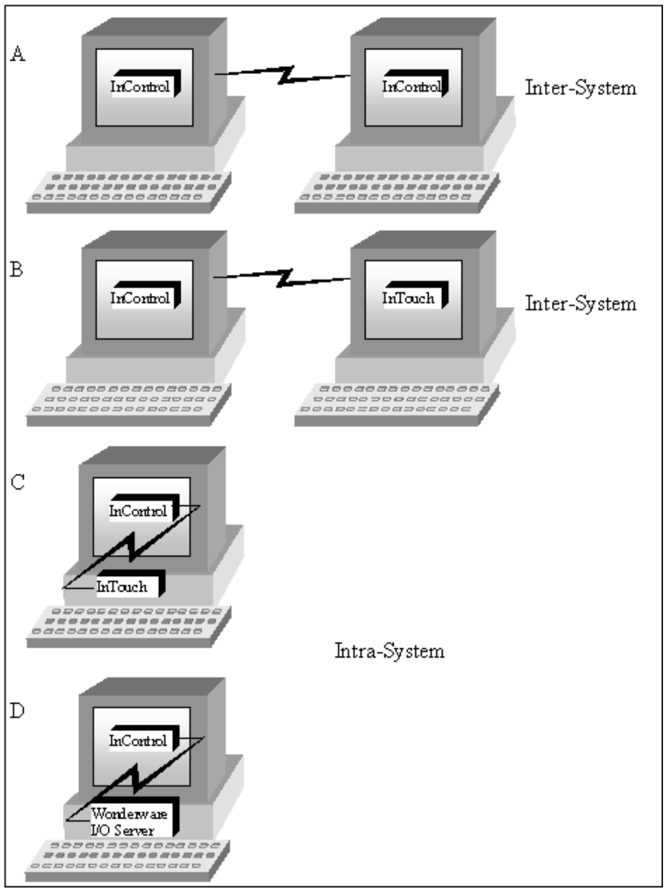
Unlike other I/O drivers, the SuiteLink driver requires no special hardware for installation. The following figure shows examples of SuiteLink connections that you can make while InControl operates as a client.

A. InControl, a server on one computer, communicates with InControl running as a client on another computer.

B. InControl, a client on one computer, communicates with InTouch running as a server on another computer.

C. InControl, operating as a client, communicates with InTouch running as a server on the same computer.

D. InControl, operating as a client, communicates with any Wonderware I/O server running on the same computer.



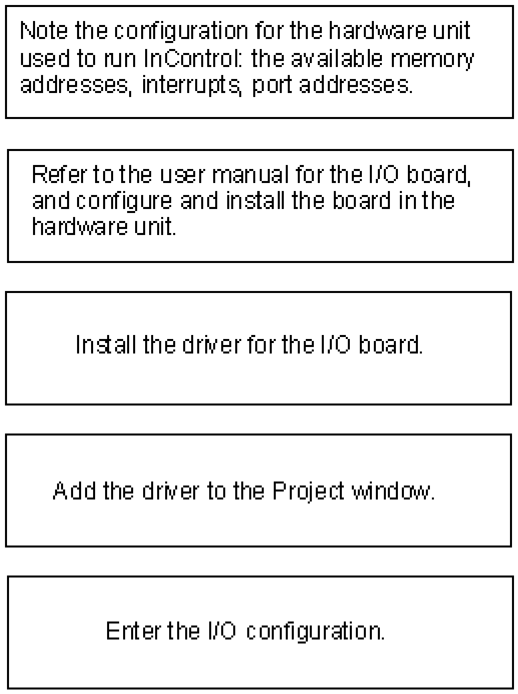
SuiteLink Connection Types

For more information about using SuiteLink, see the *Wonderware InControl SuiteLink User's Guide*.

General Installation Procedure

Various scanner board models and designs are available that allow communication to discrete and analog I/O points. For a list of the third-party I/O scanner boards currently supported by InControl, or to acquire a board, contact your distributor or the third-party manufacturer directly.

To use an I/O driver with InControl, follow this general procedure:



Note the configuration for the hardware unit used to run InControl: the available memory addresses, interrupts, port addresses.

Refer to the user manual for the I/O board, and configure and install the board in the hardware unit.

Install the driver for the I/O board.

Add the driver to the Project window.

Enter the I/O configuration.

Detailed information, such as memory offset, base, points, etc., about installation and operation of an I/O board is available in the user documentation that comes with the board. For information about configuring an individual board, refer to the I/O user guides that are copied to the hard disk when you install InControl.

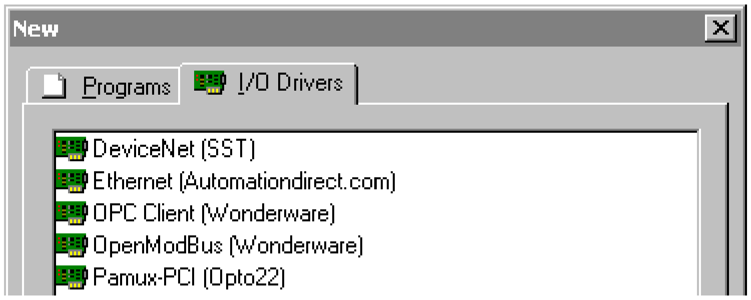
Adding/Removing Drivers

After installing the I/O board and driver, you need to add the driver to the Project window before configuring the I/O.

Adding a Driver to the Project Window

To add a driver to the Project Window:

1. Click **New** on the **File** menu. The **New** dialog box appears.
2. Click the **I/O Drivers** tab to display the list of installed drivers.



DeviceNet (SST)
Ethernet (Automationdirect.com)
OPC Client (Wonderware)
OpenModBus (Wonderware)
Pamux-PCI (Opto22)

Installed Driver List

3. Double-click the name of the driver.
4. Enter a unique name for the driver and begin configuration.

For information about configuring an individual driver, see the I/O user guides that are copied to the hard disk when you install InControl.

For additional information about configuration, see "Configuring the I/O."

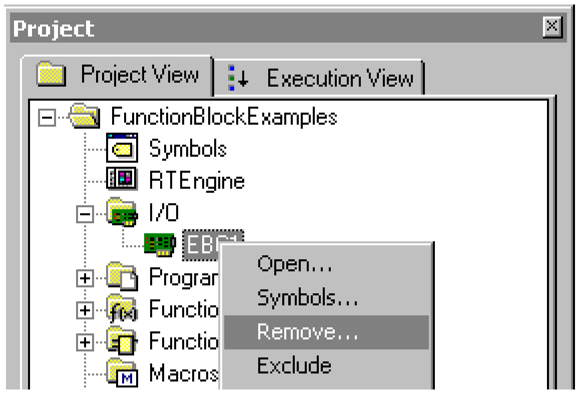
Note The Wonderware SuiteLink Client Version 2 driver is not associated with a scanner board. However, you add it to the Project window in the same way as you would any of the other drivers.

Removing or Deleting a Driver

When you remove or delete a driver configuration from the Project window, any variables that are mapped to the I/O points are deleted.

To remove or delete a driver configuration from the Project Window:

1. Select the driver and right-click as shown below.



Deleting a Driver

2. Click **Remove**. When the **Remove** dialog box appears and prompts you to confirm, click **Yes**.

   To delete the I/O driver configuration from the hard disk, check the **Delete Associated Files** checkbox in the **Remove** dialog box.

Configuring the I/O

Information about configuring the boards and assigning symbolic names to the I/O points is described in the individual I/O user guides, which are copied to the hard disk when you install InControl.

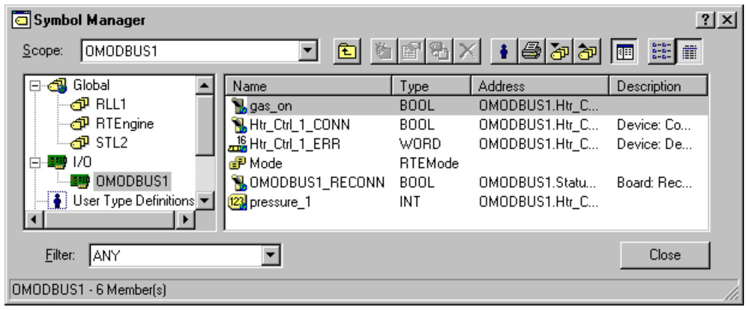
Some of the currently supported I/O drivers require a vendor-specific configuration utility for defining the I/O configuration. Check your distribution CD for these utilities and install them when using these drivers.

Some of the currently supported I/O drivers allow you to open and edit the I/O configuration of a project that will run on a remote node. The driver must be installed on both the remote node and the local node where you do the configuration. In addition, for some drivers, you may need to install the scanner board in the local node as well as in the remote node.

Note Several I/O drivers provide utilities that you can use to do an automatic configuration and/or run online diagnostics. Some of these drivers require you to use these utilities on the remote node itself. Newer drivers support remote automatic configuration and online diagnostics. To accomplish this, these drivers may download themselves to the remote runtime engine. In this case, the driver enters the Loaded mode (Mode system variable = 8).

Controller cards, control bus architectures, and I/O modules all have different timing requirements, which are often implementation specific. It is possible to set a total scan time within InControl that is faster than these devices can handle. For information about how to handle this situation, see "Adjusting the Scan Time" in the "InControl System Administration" chapter.

Driver symbols, including symbols that you generate during I/O configuration, appear in the Symbol Manager.



Name	Type	Address	Description
gas_on	BOOL	OMODBUS1.Htr_C...	
Htr_Ctrl_1_CONN	BOOL	OMODBUS1.Htr_C...	Device: Co...
Htr_Ctrl_1_ERR	WORD	OMODBUS1.Htr_C...	Device: De...
Mode	RTEMode		
OMODBUS1_RECONN	BOOL	OMODBUS1.Statu...	Board: Rec...
pressure_1	INT	OMODBUS1.Htr_C...	

OMODBUS1 - 6 Member(s)

I/O Symbols

Simulating I/O

I/O simulation provides you the following benefits as you develop a project:

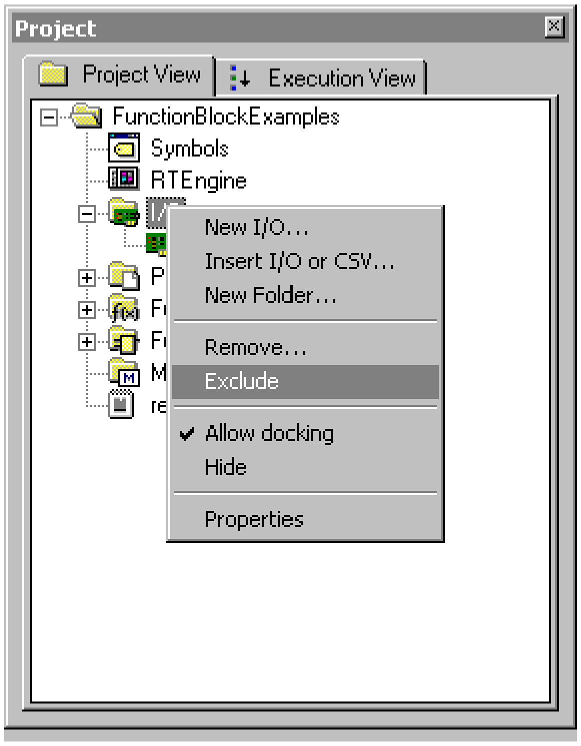
- You can define I/O tags before their associated scanner boards have been installed.
- You can test program code that references I/O tags without actually sending signals to I/O devices.

Many of the drivers that InControl supports have a **Simulate** checkbox that you can use for purposes of I/O simulation. Often this allows you to do the testing that you need. However, some third-party drivers may require the board be installed, even when they are operating in a simulation mode. Other third-party drivers may not allow you to do the testing appropriate for your application from the simulation mode.

For an alternative form of I/O simulation, you can exclude an I/O driver from the project download. If necessary, you can exclude all the drivers. Exclusion allows you to test I/O tags, even for those drivers that require that the scanner board be installed when you access the driver.

To exclude I/O tags for all I/O drivers:

1. From the Project View, right-click the I/O folder as shown below.



Excluding a Driver

2. Click **Exclude**.

To simulate I/O tags for a single driver:

1. From the Project View, right-click the driver.
2. Click **Exclude**.

C H A P T E R 5

Project Organization/ Management

This chapter describes the organization of InControl projects in terms of the IEC-61131-3 concept of the program organization unit (POU).

Contents

Overview

InControl follows the IEC 61131-3 requirement that you be able to design your code in program organization units (POUs). In this chapter, the POUs that are supported by InControl are described.

- Program
- Function Block
- Function
- Macro

You can design POUs in a variety of programming languages, including RLL, Structured Text, and SFC. Since the InControl Factory Object editor is an ActiveX container, you can also add ActiveX controls and factory objects (FOEs) to a project. The programming languages, variables, and data types that you can use in your code are described in the InControl Language Editors manual . For more information about variables and data types, see the "Defining Variables" chapter.

Programs

A program is a block of code that can be scheduled to execute automatically every scan. Programs, which are the main mechanism for executing your control logic, have the following characteristics:

- Programs execute automatically. That is, you do not have to call a program for execution, as you would call a function or function block.
- A program is defined by its programming language: RLL, Structured Text, or SFC. In addition, InControl is compatible with the ActiveX Server specification. The InControl Factory Object editor is an ActiveX container, which enables you to add ActiveX controls and factory objects (FOEs) to a project.
- Programs can contain declarations of local and global variables. The global variables can be read or written by other programs, functions, and FOEs. With a few exceptions, the local variables can be read or written only within the program where they are defined.
- InControl supports only program instances, as they are defined in the IEC 61131-3 specification. InControl does not support program type definitions.
- Programs cannot contain instances of other programs.

For information about creating programs, see "Adding a New POU to a Project."

Function Blocks

The function block POU consists of a set of programming instructions that can be called for execution by another POU. Function blocks have the following characteristics:

- InControl supports function block type definitions. You can create one or more instances of a function block type.
- A function block type is described by its programming language. Currently, InControl supports function block types that are written in Structured Text, RLL, and FBD.
- Function block instances do not execute automatically. Function block instances are executed when the code (STL, RLL, etc.) that references them is executed.
- Function block type definitions can contain declarations of local variables and input and output parameters.

  Local variables in a function block instance maintain their values between calls. This allows you to maintain a count, for example, each time the function block instance is called. However, local variables cannot be read or written by other POUs.

  Input and output parameters also maintain their values between calls. Other POUs can reference these parameters.

  The ANY data type cannot be used in a function block type definition.

  User-defined function blocks cannot be used in an SFC RLL Transition.
- Function block calls cannot be recursive. A function block instance can call another function block instance for execution, but it cannot call itself. That is, function block X cannot call itself or another function block Y, which in turn calls function block X.
- All loop constructs will continue to execute until they have completed.

For information about creating function blocks, see "Adding a New POU to a Project."

For information about defining the parameters and variables for a function block, see "Defining Function Blocks."

Functions

The function POU consists of a set of programming instructions that can be called for execution by a program, a function block, or another function. Functions have the following characteristics:

- A function is described by its programming language. Currently, InControl supports functions that are written in Structured Text, RLL, and FBD.
- Functions do not execute automatically. Functions are executed when the code (STL, RLL, etc.) that references them is executed.
- Functions can contain local variables and input and output parameters. Memory space for local variables and parameters is allocated each time the function is called, and deallocated when the function finishes execution. Variables only exist for the time that the function is executing and cannot be read or written by other POUs.
- Functions can have an optional return value that contains the result of the function after execution.

- Function calls cannot be recursive. A function can call another function for execution, but it cannot call itself. That is, function X cannot call itself or another function Y, which in turn calls function X.
- User-defined function blocks cannot be used in an SFC RLL Transition.
- All loop constructs will continue to execute until they have completed.

For information about creating functions, see "Adding a New POU to a Project."

For information about defining the parameters and variables for a function, see "Defining Functions."

Function/Function Block Differences

Three fundamental characteristics differentiate the function block and the function:

- You can create one or more instances of a function block type.
- Parameters and local variables are stored differently.

  For function block instances: local variables maintain their values between calls.

  For functions: memory space for local variables and parameters is allocated each time the function is called, and deallocated when the function finishes execution. Variables only exist for the time that the function is executing.
- Functions can have an optional return value that contains the result of the function after execution.

Macros

The macro is a specialized POU that provides a means of including one SFC, the child, for execution from a Step in another SFC, the parent. The macro, which represents an enhancement to IEC-61131-3, has the following characteristics:

- Macros are executed automatically, inline within the parent SFC.
- A macro SFC cannot include itself or its parent SFC.
- You can nest SFCs. That is, one macro SFC can call another macro SFC.
- You can call the same macro SFC from multiple points, termed Macro Steps, within an SFC, and from multiple programs.
- Macros can have local variables.

For information about creating macros, see "Adding a New POU to a Project."

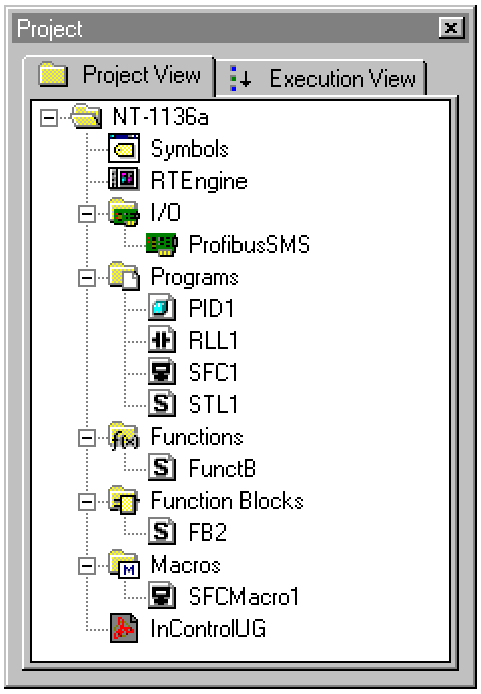
Variable Names

Use only alphanumeric characters and the underscore character for the name of a POU. Names can begin with an underscore or an alphabetic character, but not a numeric character. The maximum length of a name is as follows.

- I/O boards: 31 characters.
- Programs, Functions, Function Blocks: 31 characters.
- Projects: 255 characters.
- Other symbols: the recommended maximum length is 100 characters.

Managing Projects

All POUs that you create appear in the Project window. This window has two tabs. The Project View tab displays the programs, function block types, functions, and macros that are associated with a project. You can also handle symbol and I/O management and runtime engine configuration for a project through the Project View tab. The Execution View tab displays the priority level for programs (Normal Scan, Low Priority) and the order in which programs are executed.



The Project Window

Creating a Project

Before you can create the set of POUs, symbols, and related configuration files that handle execution of the application control logic, you must create a project.

To create a project after starting InControl:

1. On the **File** menu, click **Project**. The Project Manager appears.
2. On the **File** menu of the Project Manager, click **New**. The **Create InControl Project** dialog box appears.
3. Enter a project name using the Windows naming conventions, select a path, and click **OK**. A directory with this name is created on the hard disk and the new project is added to the list of InControl projects.
4. Double-click the project name to open the project in the Development environment. If your version of InControl supports more than one runtime engine target, the **Runtime Engine Target** dialog box appears.

5. Select the target hardware platform and click **OK**. Unless you intend to run programs on another hardware platform, select the **Windows 2000 / Windows XP / Windows 2003** target.

   The project that you created appears in the Project window of the Development environment.

Note You can convert a project that was developed for one target to run on another target. However, some I/O boards may not convert if they do not support the target operating system. See "Targeting the Hardware Platform." In addition to converting projects, you can insert program files, which are developed in a project for one target, into a project with a different target.

Removing a Project

To remove or delete a project, you must display the Project Manager.

Note You can also delete individual project files, such as programs or I/O drivers, within the InControl environment. See "Removing/Deleting a POU."

To remove or delete a project:

1. On the **File** menu, click **Project**. The Project Manager appears.
2. Select the project, and on the **File** menu of the Project Manager, click **Delete**. When the **Delete** dialog box appears and prompts you to confirm, click **Yes**.

   Note that removing a project does not delete files from the hard disk. To delete all the files associated with the project, check the **Delete Associated Files** checkbox in the **Delete** dialog box.

Adding a Project

You can add a project that you have removed from the Project Manager.

Note You can add a project (and its associated program files) only if you have not deleted it from the hard disk.

To add a project:

1. On the **File** menu, click **Project**. The Project Manager appears.
2. On the **File** menu of the Project Manager, click **Search**. The **Browse For Folder** dialog box appears.
3. Select the project and click **OK**. The project is added to the project list.

Modifying the Project Name/Description

To modify the name or description for a project:

1. On the **File** menu, click **Project**. The Project Manager appears.

2. Select the project, and on the **File** menu of the Project Manager, click **Properties**. The **Modify InControl Project** dialog box appears.
3. Enter a new description or project name, then click **OK**. The new description appears in the read-only **Description** field of the **InControl Projects** dialog box.

Adding a New POU to a Project

When you add programs, function block types, functions, or macros to a project, these items appear under their respective folders in the Project View. To organize the components of your project, you can create subfolders as needed. See "Organizing a Project."

To add a new POU to a project:

1. On the **File** menu, click **New**
2. Select the Programs tab in the **New** dialog box.
3. Select program type (Factory Object, RLL, SFC, Structured Text), and POU type (Program, Function, Function Block, Macro).
4. Choose a name (up to 31 characters).
5. If you are adding a new function block or function, you need to define parameters and variables. See "Defining Function Blocks" and "Defining Functions."

A new POU appears in a project, open and ready to edit. Double-click a POU that is closed in order to edit it.

Select an individual POU and right-click to do the following:

- **Open** Start an edit session.
- **Symbols** Access the Symbol Manager.
- **Remove** Remove the POU from the Project View.
- **Exclude** Exclude the POU from the project when you download the project. You can still download excluded POUs individually. This feature is useful for adding simulation code to your project. When a POU is excluded, the icon for the POU is dimmed.
- **Validate** Validate the code (check syntax).
- **Download** Load the POU into the runtime engine.
- **Run** Execute the POU in the runtime engine.
- **Pause** Suspend execution of the POU.
- **Stop** Stop execution of the POU and unload it from the runtime engine.

- **Properties** Check the following code properties:

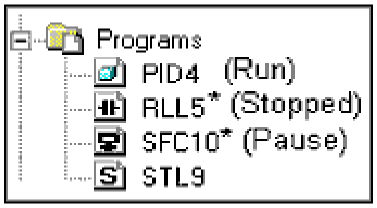
  Enter descriptive information about the POU in the **Properties** dialog box.

  Enable debugging.

  Exclude the POU from the project, described above.

  Check these dates: date the POU was modified, date the POU was compiled, and date the POU in the runtime engine was compiled.

InControl indicates the mode of a program in the Project View. The following figure shows these program modes: PID4 is running; RLL5 is stopped; SFC10 is paused (the asterisk indicates that the program has been modified and not saved, or is different from the copy in the runtime engine). Note that if mode is not shown, the program is either stopped, not loaded in the runtime engine, the Development environment is not connected to the runtime engine, or the project is different from the copy in the runtime engine.



Programs
PID4 (Run)
RLL5* (Stopped)
SFC10* (Pause)
STL9

InControl indicates that functions and function blocks are loaded in the runtime engine by displaying their mode in the Project View as "Loaded."

Adding an Existing POU to a Project

To add an existing POU to a project:

1. On the **Insert** menu, click **Files into Project**. The **Insert Files Into Project** dialog box appears.
2. Locate the POU that you want to add.
3. Click **OK**. The POU is added to the Project View.

Note All POUs are inserted under the Programs folder of the Project window. You must move functions to the Functions folder, function block types to the Function Block folder, and macros to the Macros folder for the project to compile correctly.

You can add non-InControl program files to the project, such as, a Readme.txt file.

Double-clicking Readme.txt in the Project View opens the Windows Notepad utility and displays the Readme.txt file.

If you add an executable file, such as the Windows Wordpad utility (wordpad.exe), double-click the file name in the Project View to run the executable file.

Note You can add a file to a project by dragging and dropping it from any application that supports the Drag and Drop feature, such as the Windows Explorer.

Removing/Deleting a POU

To remove or delete a POU from a project, you must display the Program View.

To remove or delete a POU from a project:

1. Display the Project View.
2. Select the POU and right-click as shown below.
3. Click **Remove**. When the **Remove** dialog box appears and prompts you to confirm, click **Yes**.

   Note that removing a POU does not delete files from the hard disk. To delete the associated files, check the **Delete Associated Files** checkbox in the **Remove** dialog box.

Renaming a POU

To rename a POU, use the **Save As** option on the **File** menu.

To rename a POU:

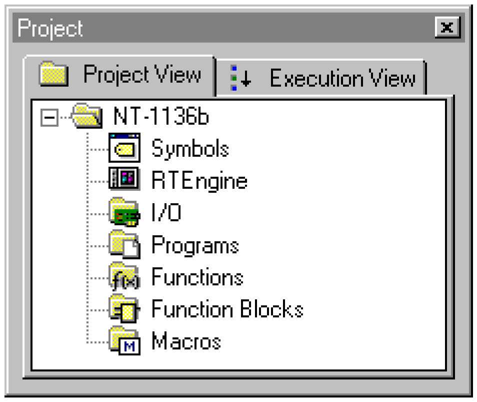
1. Display the Project View.
2. Select the POU.
3. Click **Save As** on the **File** menu.
4. Choose a name (up to 31 characters). A message box prompts you to confirm whether to add the POU to the project.

Note The original POU remains in the project after you save it with a different name.

Organizing a Project

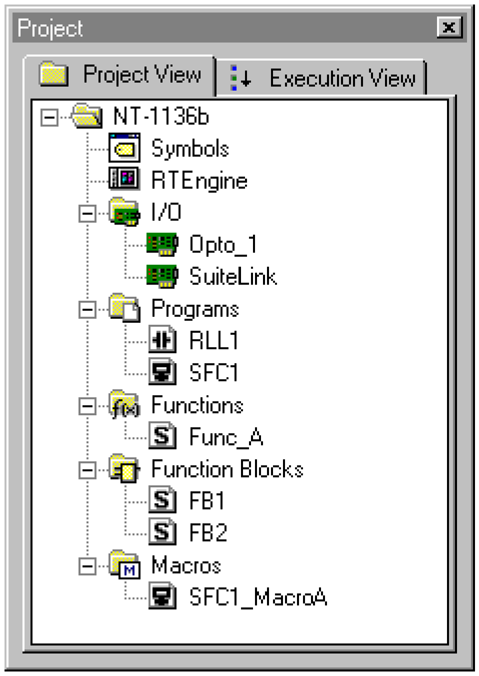
The InControl Project window is a versatile tool that allows you easy organization for the components of your project.

The following figure shows the default folder arrangement in a Project window of a new project.



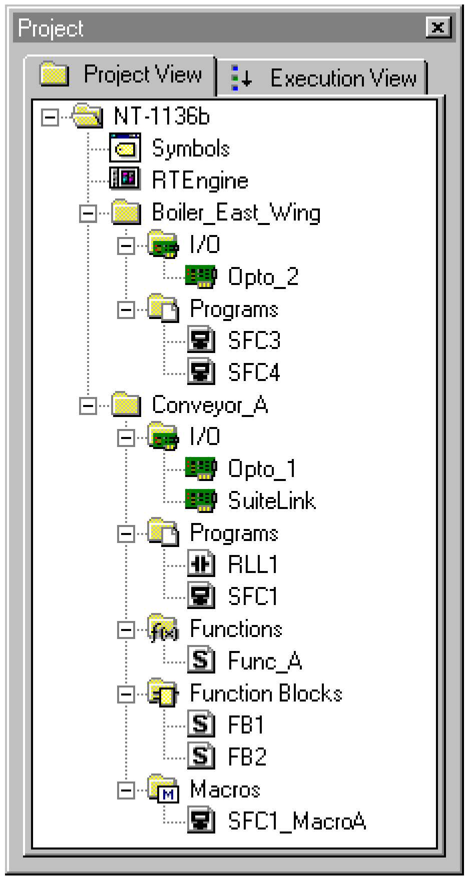
Project Window - Default Organization

When you add POUs and I/O configurations the new objects appear in the appropriate folders. The default organization is to group together all the I/O configurations, all the programs, all the function block types, etc. In the following figure, the I/O folder displays two I/O configurations, the Programs folder displays two programs, the Functions folder displays one function, the Function Blocks folder displays two function block types, and the Macros folder displays one macro.



Project Window - Folder Organization

If you want to group POUs and I/O by unit areas of the factory floor, you can right-click the project name and create a new folder. Right-click the new folder and add the POUs and I/O configurations that are appropriate for that factory unit. In the following figure, two I/O configurations, two programs, one function, two function block types, and one macro appear under the Conveyor_A folder. The other factory unit, a boiler with a folder called Boiler_East_Wing, is represented by one I/O configuration and two programs.



Project Window - Area Organization

You can drag existing POUs and I/O configurations to folders if you want to rearrange the project organization. If the appropriate subfolder does not exist (you drag a function into a folder that does not contain a Function folder already), InControl creates the subfolder automatically.

You can rename any of the folders, including the standard project folders (Programs, Functions, Macros, etc.).

Defining Function Blocks

After you create a new function block type, you need to specify its parameters and variables.

See the *InControl Language Editors* manual for examples that show how to develop user-defined function blocks.

Setting Parameters and Variables

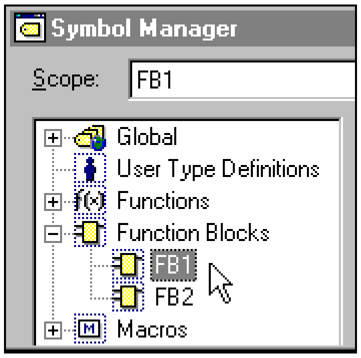
You define the input and output parameters and variables for a function block type in the Symbol Manager. Function block local variables are local to the function block instance and cannot be referenced by any other POU. Within the function block instance, input parameters are read only. Output parameters must be assigned values through an assignment statement.

Note these guidelines when you develop a function block in RLL.

- You can define up to eight input or input-output (InOut) parameters. It is recommended that you define a Boolean output (the .ENO) as the first output parameter.
- You can define up to eight output parameters. It is recommended that you define a Boolean output (the .ENO) as the first output parameter.
- In RLL, function blocks run on every scan, regardles of the state of the rung input. It is recommended that you create a Boolean input (the .EN) and output (the .ENO) as the first parameters to manage the rung logic. Assign the value of the .ENO to the .EN.

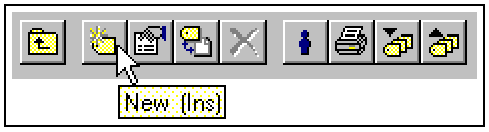
To define a parameter or variable for a function block type:

1. Add a function block type to the project. For information about creating function blocks, see "Adding a New POU to a Project."
2. On the **Tools** menu, click **Symbol Manager**.
3. Click the function block type to select it as shown in the following figure.

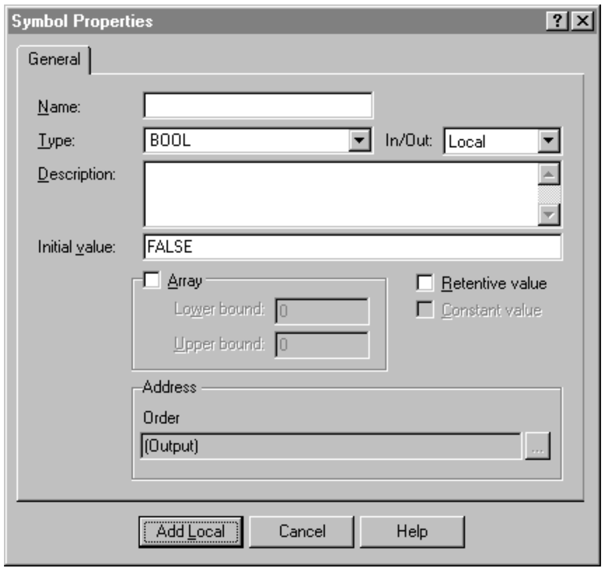


Selecting Function Block Type

4. Click **New** on the Symbol Manager toolbar.



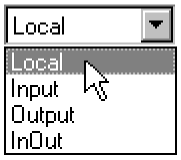
The **Symbol Properties** dialog box appears.



Name:
Type: BOOL
In/Out: Local
Description:
Initial value: FALSE
Array: [ ]
Lower bound: 0
Upper bound: 0
Retentive value: [ ]
Constant value: [ ]
Address
Order
(Output)

Function Block - Symbol Properties

5. Enter the name of the parameter or variable into the **Name** field. Use only alphanumeric characters and the underscore character.
6. Select the data type in the **Type** field.
7. Enter the optional description into the **Description** field.
8. Choose variable or type of symbol in the **In/Out** field, as shown below.



9. If the parameter or variable requires an initial value, enter the value into the **Initial Value** field.
10. Click the **Retentive Value** checkbox if the variable is to retain its value in the event of a power loss.

    InControl periodically saves retentive variables to the hard disk. The default frequency is zero. This interval is configurable and you can set it in the Runtime Engine Properties dialog box, described in "Setting Scan Times" of the "InControl System Administration" chapter.
11. If you are defining the parameter or variable as an array, check the **Array** checkbox. This causes the **Lower Bound** and **Upper Bound** fields to become active. Enter the lower and upper values in the appropriate fields.
12. Click **Add Local** to complete the definition for the parameter or variable.

Note You can change the order of the parameters in an existing function block. Access the function block type in the Symbol Manager. Right-click the parameter and click the **Decrease Address** or **Increase Address** options.

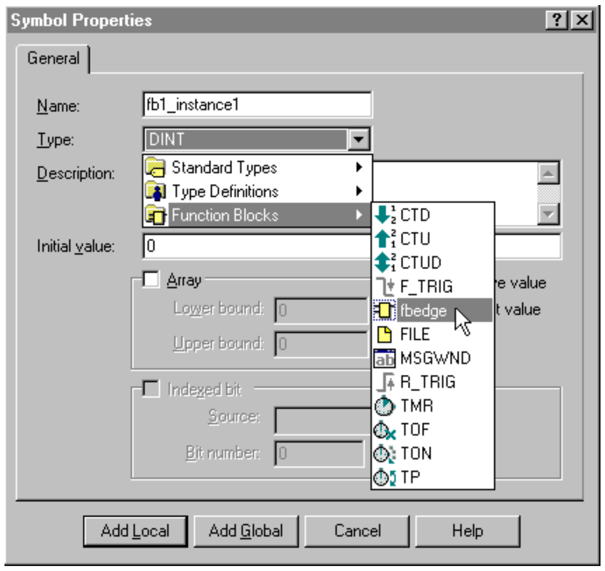
Defining an Instance

You define an instance of a function block type in the Symbol Manager.

To define an instance of a function block:

1. Access the Symbol Manager.
2. Select the appropriate scope, for example, global, an individual program, another function block, etc.
3. Click **New** on the Symbol Manager toolbar.

   The **Symbol Properties** dialog box appears.
4. Enter the name of the instance into the **Name** field. Use only alphanumeric characters and the underscore character.
5. Select the function block type in the **Type** field.



Symbol Properties

Name: fb1_instance1

Type: DINT

Initial value: 0

Selecting Function Block Type

6. Enter the optional description into the **Description** field.
7. Since all the remaining properties are defined by the function block type, click **Add Local** or **Add Global** to complete the definition for the instance.

Entering Code for the Call

You can call a function block instance for execution from any other POU that is written in the Structured Text language. This section gives two examples of the syntax you can use.

```
<FB_Instance> (<Parameter1Value>, <Parameter2Value>,
    <Parameter3Value>,..);<result> :=
    (<FB_Instance.ParameterOut>);
```

For example, you have defined a function block type called CALC that does a calculation based on two input parameters called IN1 and IN2 and that writes to one output parameter called OUT. You create an instance called CALC1 and you want to write the result to a variable called CALCRESULT. The syntax is the following:

```
CALC1 (IN1:= 55.55,IN2:=66.66);CALCRESULT := CALC1.OUT;
```

The following code gives the same result:

```
<FB_Instance.ParameterIn1> := <Value1>;
<FB_Instance.ParameterIn2> := <Value2>;
<FB_Instance.ParameterIn3> := <Value3>;
<FB_Instance> ( );
<result> := (<FB_Instance.ParameterOut>);
```

Using the same example given above, the syntax is the following:

```
CALC1.IN1:= 55.55
CALC1.IN2:= 66.66
CALC ();
CALCRESULT := CALC1.OUT;
```

Defining Functions

After you create a new function, you need to specify its parameters and variables. If the function returns a value, you need to specify its data type.

See the *InControl Language Editors* manual for examples that show how to develop user-defined functions.

Setting Parameters and Variables

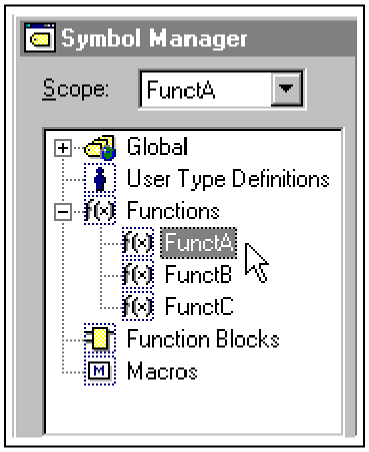
You define the input and output parameters and variables for a function in the Symbol Manager. Function variables are local to the function and cannot be referenced by any other POU, except within the context of the function call. Within the function, input parameters are read only. Output parameters must be assigned values through an assignment statement.

Note these guidelines when you develop a function in RLL.

- You can define up to seven input or input-output (InOut) parameters. The editor always adds an eighth input by default, which acts as the EN input.
- You can define up to seven output parameters. The editor always adds an eighth output by default, which acts as the ENO output.
- If you define a Boolean output parameter, the state of this output determines the output state of the rung, which contains the function, in the calling program.

To define a parameter or variable for a function:

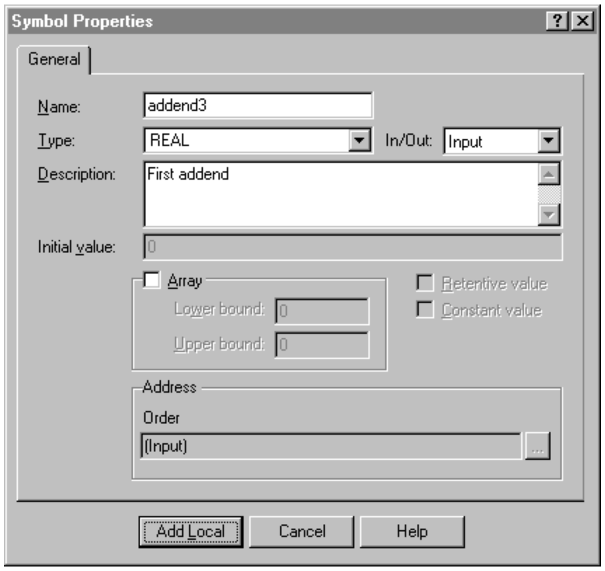
1. Add a function to the project. For information about creating functions, see "Adding a New POU to a Project."
2. On the **Tools** menu, click **Symbol Manager**.
3. Click the function to select it as shown in the following figure.



Selecting Function

4. Click **New** on the Symbol Manager toolbar.

   The **Symbol Properties** dialog box appears.



Symbol Properties

General

Name: addend3

Type: REAL

In/Out: Input

Description: First addend

Initial value: 0

☐ Array

Lower bound: 0

Upper bound: 0

☐ Retentive value

☐ Constant value

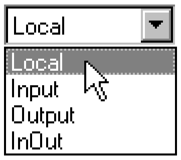
Address

Order

(Input)

Function - Symbol Properties

5. Enter the name of the parameter or variable into the **Name** field. Use only alphanumeric characters and the underscore character.
6. Select the data type in the **Type** field.
7. Enter the optional description into the **Description** field.
8. Choose variable or type of symbol in the **In/Out** field, as shown below.



9. If the parameter or variable requires an initial value, enter the value into the **Initial Value** field.
10. If you are defining the parameter or variable as an array, check the **Array** checkbox. This causes the **Lower Bound** and **Upper Bound** fields to become active. Enter the lower and upper values in the appropriate fields.
11. Click **Add Local** to complete the definition for the parameter or variable.

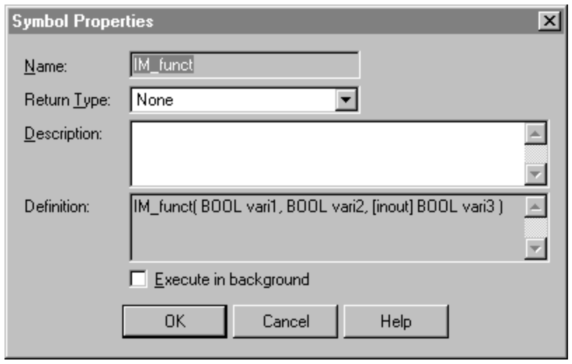
Note You can change the order of the parameters in an existing function. Access the function in the Symbol Manager. Right-click the parameter and click the **Decrease Address** or **Increase Address** options.

Specifying Data Type for a Function Return Value

You specify the return type for a function in the Symbol Manager.

To specify the return data type for a function:

1. Access the Symbol Manager.
2. Right-click the function and select **Properties**. The **Symbol Properties** dialog box appears.



Name: IM_funct
Return Type: None
Definition: IM_funct(BOOL vari1, BOOL vari2, [inout] BOOL vari3)

Function – Symbol Properties Return Value

3. For a function, select the data type in the **Return Type** field. For a procedure, select **None** as the Return Type.

Functions only return simple data types. They cannot return arrays, structures, or function blocks.

4. Select **Execute in Background** if appropriate, then click **OK**. For information about background execution, see "Functions and Background Execution."

Functions and Background Execution

For any functions that require a significant period of time to execute it is recommended that you configure them to run in background when possible. When a function runs in background, the scan is not delayed while the function completes execution.

- For a function call made from an STL program:
  As the function runs in background, the next program in the project's order of execution runs. If the function has completed by the next scan, program flow continues at the line of code following the function call. Otherwise, execution for the calling program continues to wait for the completion of the function. No other lines of code in the program are executed, and the next program in the project's order of execution continues.
- For a function call made from an RLL program:
  As the function runs in background, no more logic is solved on the rung with the function call. Logic on the following rungs is solved, however. In the next scan, logic on all rungs preceding the rung with the function call is solved. If the function has completed, the remaining logic on the rung with the function call is solved. Otherwise, program flow continues on the subsequent rungs.

Within a program, only one function can be running in background at a time. If more than one background function call is made from a program, the subsequent functions wait until the function that is executing has finished.

When a function runs in background, program flow is not paused at breakpoints. All loop-type constructs operate as if they have a terminating End-No-Wait.

Entering Code for the Call

You can call a function for execution from any other POU that is written in the Structured Text language.

A function that does not return a value operates like a procedure. The format for this type of function is the following:

```
<Function> (<Parameter1Value>, <Parameter2Value>,
    <Parameter3Value>,..);
```

For example, you have defined a function called CALC that does a calculation based on two input parameters called IN1 and IN2 and that writes to one output parameter called OUT. The syntax is the following:

```
CALC (IN1:= 55.55,IN2:= 66.66, OUT:= CALCRESULT);
```

You can specify a return value for a function. A function that returns a value operates as a true function and you use it on the right side of an Assignment statement. The format for this type of function is the following:

```
<result> := <Function> (<Parameter1Value>,
    <Parameter2Value>,..);
```

Set the return value by including code in the function that assigns the value to the function name.

For example, you have defined a function called CALC that does a calculation based on two input parameters called IN1 and IN2 and that returns a value of data type LREAL. The syntax of the function call is the following:

```
CALCRESULT := CALC (IN1:= 55.55,IN2:= 66.66);
```

The actual code in the function is the following:

```
CALC := IN1*IN2;
```

Note how you set the return value by placing the function name CALC on the left side of the Assignment statement.

Alternative forms for the syntax of a function call allow you to use assignment statements for parameters, or to list variables or literal values for parameters in the order defined by their addresses in the Symbol Manager.

In the following syntax, parameters receive their values through assignment statements. In this form, parameters can be in any order.

```
(<Parameter1:= Value1>, <Parameter2:= Value2>,
   <Parameter3:= Value3>,...);
```

In the following example, parameters appear in any order.

```
CALC (OUT:= CALCRESULT, IN2:= 5.5,IN1:= 6.6);
```

In the following syntax, parameters receive their values according to the parameter order in the Symbol Manager.

```
(<Value1>, <Value2>, <Value3>,...);
```

In the following examples, parameter order is fixed.

```
CALC (55.55, 66.66, CALCRESULT);
```

You must enter all parameters. You cannot skip a parameter by using an extra comma. You must use the same syntax for all parameters used in the function. That is, do not use an assignment statement for one parameter and a variable or literal value for another.

Adding/Organizing I/O Drivers

When you add I/O drivers to the project, the drivers appear under the I/O folder in the Project View. To organize the drivers, you can create folders for various drivers under the I/O folder and place drivers within them as needed. Click the I/O folder and right-click. Select **New Folder** to create a folder under the I/O folder.

To add an I/O Driver to a project:

1. Click **New** on the **File** menu.
2. Select the I/O Drivers tab in the **New** dialog box.
3. Select the I/O Driver and click **OK**.

For detailed instructions that describe adding or removing drivers, see the "I/O Configuration" chapter.

Double-click an I/O driver to configure it. Select a driver and right-click to enter descriptive information about the driver in the **Properties** dialog box.

You can also right-click to configure or remove a driver, or to exclude it from the project when you download the project.

When you exclude an I/O driver, the I/O symbols are downloaded, but the driver does not execute. This feature is useful for simulating I/O without having I/O hardware physically present.

Note Removing an I/O configuration does not delete files from the hard disk. To delete the I/O driver configuration, check the **Delete Associated Files** checkbox in the **Remove** dialog box.
When you remove or delete an I/O driver configuration from the Project View, any variables that are mapped to the I/O points are deleted.

Configuring the Runtime Engine

You can set scan time and other runtime parameters, such as automatic startup for the runtime engine from the Project View. Double-click the RTEngine icon in the Project View and the dialog box for the runtime engine appears.

Right-click the RTEngine icon in the Project View to do the following:

- **Connect/Disconnect** Connect/disconnect the Development environment and the runtime engine.
- **Configure** Display the **Offline Runtime Engine Properties** dialog box if the Development environment not connected to the runtime engine. Display the **Online Runtime Engine Properties** dialog box if the Development environment is connected to the runtime engine.
- **Report Status** Examine runtime engine status data, such as current project, time stamp, scan time, mode, processor utilization, faulted programs, I/O faults, etc. This data appears in the Output window and the Wonderware Logger.
- **Clear Faults** Set faulted programs to Pause mode, clear I/O faults, and clear runtime engine error status bits, such as RTEngine.ScanOverrun.
- **Pause** Set the runtime engine to the Pause mode.
- **Single Scan** Execute a single scan of the runtime engine.
- **Stop** Set the runtime engine to the Stop mode.
- **Properties** Check the following code properties:

Enter descriptive information about the runtime engine in the **Properties** dialog box.

Change the runtime engine target. This is useful when you develop and test a project on a computer using the Windows operating system but intend to download and run the project on a computer that uses another operating system.

Check these dates: date the runtime engine was modified, and date that a project was downloaded to the runtime engine.

For detailed information about the runtime engine, see these chapters:

"Running a Project" and "InControl System Administration."

Accessing the Symbol Manager

Double-click the Symbols icon in the Project View to open the Symbol Manager. You create or edit POU variables in the Symbol Manager. Right-click the Symbols icon to enter descriptive information about your symbol configuration in the **Properties** dialog box. The Symbol Manager is shown in the following figure.

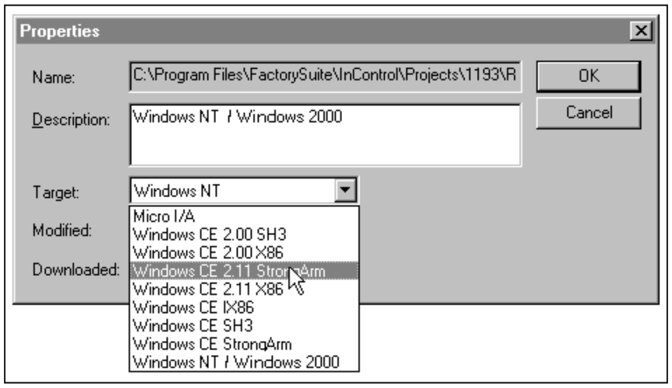
For detailed information about the Symbol Manager, see the "Defining Variables" chapter.

Targeting the Hardware Platform

Some versions of InControl support multiple runtime hardware platforms. If you have purchased one of these versions, then when you create a new project, InControl prompts you to select the platform (the target) where you intend to run the project. This allows you to develop a project in the Windows environment, and then download and run the project on any of several hardware platforms. You can convert a project that was developed for one target to run on another target.

To change the target runtime engine:

1. Select the runtime engine icon in the Project View and right-click. The **Properties** dialog box for the runtime engine appears.



Properties

Name: C:\Program Files\FactorySuite\InControl\Projects\1193\R

Description: Windows NT / Windows 2000

Target: Windows NT

Modified:

Downloaded:

Targeting the Hardware Platform

2. The current target is shown in the **Target** field. Select the new target and click **OK**.

Note Some I/O boards may not convert if they do not support the target operating system. After the conversion, these boards are dimmed in the Project window.

Changing Program Priority and Execution Order

Use the Execution View to specify the priority level for programs (Normal Scan, Low Priority), and to change the order within the scan in which they are executed. If you do not set the order, programs are executed in the order in which you create them.

For more information about priority level, see "Runtime Engine Timeline" in the InControl System Administration chapter.

For detailed information about changing priority and execution order, see "Project/Program Execution Order" in the Running a Project chapter.

C H A P T E R 6

Defining Variables

This chapter describes how to define the symbols that you use in your application program.

Contents

- Editing Tips - Copy / Paste / Move Symbols
- Creating a Variable
- Creating an Array of Variables
- Referencing Arrays
- Assigning a Name to a Bit in a Variable
- Creating a User-Defined Data Type
- Printing Information for Variables
- System Variables - General
- System Variables - Runtime Engine
- Transferring Symbol Databases
- Symbol Exchange Between InControl and InTouch

Introduction

The InControl programming tools allow you to define and use variables, which are internal memory locations that contain project data. The content of the information is defined by the data type and can be real numbers, integers, strings of characters, etc. Use the Symbol Manager to define a variable, assign it a symbolic name and data type, and designate its scope as local or global.

Variable Names

Use only alphanumeric characters and the underscore character for the name of a variable. Names can begin with an underscore or an alphabetic character, but not a numeric character. The recommended maximum length of a variable name is 100 characters.

Local and Global Variables

Variables can be local or global in scope.

- Global variable A global variable can be used and referenced within a project by all programs, including InControl factory objects (FOEs). You can use global variables within SuiteLink and DDE operations, and you can reference I/O points as global variables.
- Local variable A local variable is used and referenced only within the program in which it is defined. Except for local variables defined for functions, you can reference a local variable through SuiteLink and DDE operations. To help coordinate program execution, you can reference the following local system variables in other programs:

  Mode: Contains the current mode of a program. Use this syntax to reference Mode:
  `<program name> . Mode`

  FOE variables: FOEs and other ActiveX controls add local variables automatically when they are installed. You cannot edit these variables from within the Symbol Manager. Use this syntax to reference FOE variables:
  `<FOE name> . <variable name>`

For information about using DDE to reference local and global variables, see the "Monitoring Data By DDE/SuiteLink" appendix.

For information about using SuiteLink to reference local and global variables, see the *Wonderware InControl SuiteLink User's Guide*.

In contrast to variables, which are internal memory locations, I/O points are external locations, associated with physical points. Because you can reference them in a program just like variables, I/O points appear in the Symbol Manager and are listed under the I/O group. I/O points are scoped like global variables: you can reference them from any program, and you can use them in SuiteLink operations. You cannot edit an I/O point from within the Symbol Manager.

Variables Assigned a Constant Value

Assign a constant value to a variable when you need a variable, such as pi (p), that has an unchanging literal value to be used in your program. For some variables, such as a recipe ingredient, you may want to test a number of values until you determine the one to use for the variable. In this case, you can access the **Symbol Properties** dialog box for the variable, assign an initial value based on your tests, and then check the **Constant Value** checkbox.

- If you add a variable that has been defined as a constant to the Watch window, the variable appears as a dimmed value. You cannot modify the variable in the Watch window.
- You cannot directly monitor a variable, which has a constant value, from an external application, such as InTouch. Assign the value of the variable to another variable that is not constant, and then monitor the second variable.

Check the **Constant Value** checkbox on a variable's **Symbol Properties** dialog box to if you want to define a named variable and give it a constant value. The **Symbol Properties** dialog box is described in "Creating a Variable."

Retentive Variables

You can define a variable as retentive if you want the option of backing up the value of a variable to the hard disk. InControl provides three ways by which you can specify for the backup to occur.

- If the runtime engine shuts down during a power failure, the value of a retentive variable is copied to the hard disk. Note that the values of any variables that have been forced are also saved during a power failure.

  The values of retentive and forced variables are not saved unless you are using an intelligent UPS with the system and you have configured it to signal InControl of the power failure. For more information about preparing for power failures, see "Handling Power Failure" in the "InControl System Administration" chapter.

- You can configure InControl to save retentive and forced variables to the hard disk periodically. The default frequency of zero disables this feature. You can change it in the **Runtime Engine Properties** dialog box, described in "Setting Scan Times" of the "InControl System Administration" chapter.

  Take into account the number of retentive variables in your project when you choose the frequency of the update. A short update interval can degrade the performance of your system when a large number of the variables are marked as retentive.

- You can design code in a program to save the value of retentive and forced variables on demand. For a forced variable, both the value and the forced state are saved to the hard disk. Use the following syntax:

```
RTEngine.ExecProjectCommand (SaveRetentive);
```

The values are only restored when the runtime engine is configured to restart automatically (Last, Pause, Run mode) after a system reboot. For more information, see "Restarting Projects Automatically" in the "InControl System Administration" chapter.

Check the **Retentive Value** checkbox on a variable's **Symbol Properties** dialog box to if you want the option of backing up the value of a variable to the hard disk. The **Symbol Properties** dialog box is described in "Creating a Variable."

For more information about the behavior of variables at runtime, see "Variables and Runtime Operation."

Enumerated Variables

An enumerated variable is a type of data structure, the members of which are a set of DINT data types. Use an enumeration when you need to define a group of named constants. When logic that contains the constant is solved, the value of the constant is the initial value, which you assign when you define the constant.

For a simple Structured Text example that uses an enumeration, see the CASE statement in the Structured Text Language chapter. For instructions that explain how to define an enumeration, see Creating a User-Defined Data Type."

Read-Only Variables

Some variables, (including those belonging to objects such as the runtime engine, FOEs, programs, and function blocks), are read only at runtime. This means that the value of the variable is determined by the object. Any changes that you make at runtime through the Watch window or a program, for example, are always overwritten by the object.

For more information about the behavior of variables at runtime, see "Variables and Runtime Operation."

Forced Variables

At runtime, you can force the value of a variable to a particular value. This means that the value does not change as the program runs until you force it to a new value or unforce it.

For more information about the behavior of variables at runtime, see "Variables and Runtime Operation."

Variables and Runtime Operation

This section describes the behavior of variables at runtime.

Read-Only and Read-Write Variables

Values of read-only variables are determined by their associated object, such as an FOE or the runtime engine, for example. This means that you cannot change read-only variables through the Watch window, an HMI, a program, through the Symbol Manager, or a program editor. If you do attempt to change a read-only variable at runtime, any value that you write will be overwritten.

Retentive Variables

If you redownload an object (such as an FOE or program) after a system restart that caused the values of retentive variables to be used, then the retentive values will be overwritten by any initial values that you defined when you created the object.

For example, you define the setpoint of a PID to be 50 degrees. During operation by the PID, the setpoint has been changed to 30 degrees. If you have provided for the back-up of retentive variables (described in "Retentive Variables" then following a power cycle, the value of 30 degrees is used for the setpoint. If you later redownload the PID, however, 50 degrees is used for the setpoint.

Uploadable Variables

If you upload values from an object (such as a PID) at runtime, then the uploaded values overwrite the ones used when you initially created the object. If you later redownload the object, the values downloaded with the object are the ones that were uploaded previously.

You can mark uploadable values as being retentive. In the PID setpoint example above, this means that the setpoint will be 30 degrees, not 50 degrees, following a redownload of the PID. This assumes that you have provided for the backup of retentive variables as described in "Retentive Variables."

Forced Variables

When a variable has been forced at runtime, the forced value is always used for any calculations based upon the variable or its display in an HMI or the Watch window. The value does not change as the project runs until you force the variable to a new value or unforce it.

Variable Data Type Groups

The following table lists the IEC 61131-3 data types that are supported by InControl. Individual data types are described in the pages that follow.

Data Types and Categories

All Types	Group	SubGroup	Data Type
ANY	ANY_NUM	ANY_REAL	LREAL
			REAL
		ANY_INT	DINT
			INT
			SINT
			DWORD [1]
			WORD [1]
			BYTE [1]
ANY	ANY_BIT		DWORD
			WORD
			BYTE
			BOOL
	ANY_DATE		DT (date and time)
			DATE
			TOD
			TIME
			TMR [2]
			FILE [2]
			STRING
			User-Defined
			ANY
Enumeration			RTEMODE [2]

Note The LINT, ULINT, and LWORD data types are not currently supported by InControl.

1 The UDINT, UINT, and USINT data types are equivalent to the DWORD, WORD and BYTE data types respectively. An InControl enhancement to the ANY_BIT data types makes the UDINT, UINT, and USINT data types unnecessary.

2 Enhancement to the IEC 61131-3 specification.

WARNING! IEC-61131 does not support the combination of signed and unsigned numbers (ANY_NUM data types) in a math calculation. If you do combine signed and unsigned numbers, the results of the math operation may not be what you expect, which may have the potential risk of death or injury to personnel and/or damage to equipment. Avoid using expressions that combine signed and unsigned numbers.

LREAL

The LREAL data type is a member of the ANY_REAL group of data types. LREAL data types are valid in any instruction or function block that accepts an ANY, ANY_NUM, ANY_REAL, or LREAL data type. An LREAL number data type is a 64-bit value composed of one or more of the digits (0-9), is signed, and contains a decimal point. The range for LREAL numbers is the following: -1.79769313486231 E308 (negative) to +1.79769313486231 E308 (positive), and includes zero. The IEEE format is used to represent LREAL data types.

Note When you communicate with the runtime engine using a SuiteLink/DDE interface, 64-bit LREAL data types are transmitted at 32-bit precision.

REAL

The REAL data type is a member of the ANY_REAL group of data types. REAL data types are valid in any instruction or function block that accepts an ANY, ANY_NUM, ANY_REAL, or REAL data type. A REAL number data type is a 32-bit value composed of one or more of the digits (0-9), is signed, and contains a decimal point. The range for REAL numbers is the following: -3.402823 E38 (negative), to +3.402823 E38 (positive), and includes zero. The IEEE format is used to represent REAL data types.

DINT

The DINT data type is a member of the ANY_INT group of data types. DINT data types are valid in any instruction or function block that accepts an ANY, ANY_NUM, ANY_INT, or DINT data type. The DINT is a signed integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. The DINT is 32 bits in length and has a range of -2147483648 to +2147483647.

INT

The INT data type is a member of the ANY_INT group of data types. INT data types are valid in any instruction or function block that accepts an ANY, ANY_NUM, ANY_INT, or INT data type. The INT is a signed integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. The INT is 16 bits in length and has a range of -32768 to +32767.

SINT

The SINT data type is a member of the ANY_INT group of data types. INT data types are valid in any instruction or function block that accepts an ANY, ANY_NUM, ANY_INT, or SINT data type. The SINT is a short signed integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. The SINT is 8 bits in length and has a range of -128 to +127.

Unsigned Integers

The UDINT, UINT, and USINT data types are equivalent to the DWORD, WORD and BYTE data types respectively. An InControl enhancement to the ANY_BIT data types makes the UDINT, UINT, and USINT data types unnecessary.

Note that you can use number bases other than ten for literal numbers that are ANY_INT data types. Use these formats: 2#**<num>**, 8#**<num>**, 16#**<num>**.

You can reference an individual bit within a BYTE, WORD, or DWORD variable. See "Assigning a Name to a Bit in a Variable" for more information.

DWORD

The DWORD data type is a member of the ANY_BIT and ANY_INT groups of data types. DWORD data types are valid in any instruction or function block that accepts an ANY, ANY_BIT, or DWORD data type. A DWORD is an unsigned integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. A DWORD is 32 bits in length and has a range of 0 to 4294967295.

WORD

The WORD data type is a member of the ANY_BIT and ANY_INT groups of data types. WORD data types are valid in any instruction or function block that accepts an ANY, ANY_BIT, or WORD data type. A WORD is an unsigned integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. A WORD is 16 bits in length and has a range of 0 to 65535.

BYTE

The BYTE data type is a member of the ANY_BIT and ANY_INT groups of data types. BYTE data types are valid in any instruction or function block that accepts an ANY, ANY_BIT, or BYTE data type. A BYTE is an unsigned integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. A BYTE is 8 bits in length and has a range of 0 to 255.

BOOL

The BOOL data type is a member of the ANY_BIT group of data types. BOOL data types are valid in any instruction or function block that accepts an ANY, ANY_BIT, or BOOL data type. A BOOL is one bit in length and can have one of two values: TRUE (1, or on) or FALSE (0, or off).

Date / Time Data Types

The DATE, DT, and TOD data types are eight-bit floating point numbers. The value of the day is represented by a whole number with midnight of December 30, 1899 equal to zero. The value of an hour is the absolute value of the fractional part of the number. See examples in the following table.

Date / Time Format Examples

Date and Time	Value
Midnight, December 30, 1899	0.00
Midnight, January 1, 1900	2.00
6:00 A.M. January 4, 1900	5.25
Noon, January 4, 1900	5.5
9:00 P.M. January 4, 1900	5.875

DT

The DT data type is a member of the ANY_DATE group of data types. DT data types are valid in any instruction or function block that accepts an ANY, ANY_DATE, or DT data type.

The DT data type has the following format:
DATE_AND_TIME | date_and_time | DT | dt#YYYY-MM-DD-HH:MM:S.S, where YYYY (100-2100) is the year, MM (1-12) is the month, DD (1-31) is the day of the month, HH (0-23) is the hour, MM (0-59) is the minute, and S.S (0.0-59.0) is a real number containing seconds.

If you create an expression of DT data types, inputs and outputs must be the data types listed in "Using Date/Time-Based Data Types in Expressions."

The TODAY and NOW system variables, described in "DATE" and "TOD" can also be used as a DT data type.

DATE

The DATE data type is a member of the ANY_DATE group of data types. DATE data types are valid in any instruction or function block that accepts an ANY, ANY_DATE, or DATE data type.

The DATE data type has the following format:
DATE | date | D | d#YYYY-MM-DD, where YYYY (100-2100) is the year, MM (1-12) is the month, and DD (1-31) is the day of the month.

If you create an expression of DATE data types, inputs and outputs must be the data types listed in "Using Date/Time-Based Data Types in Expressions."

The TODAY system variable is a DATE data type that contains the current system date and can be used to determine when an event takes place. These operators can be used with TODAY: EQ, LT, GT, LE, GE, and NE. To read the value of TODAY, use the assignment statement or MOVE command to move the value to a variable.

TOD

The TOD data type is a member of the ANY_DATE group of data types. TOD data types are valid in any instruction or function block that accepts an ANY, ANY_DATE, or TOD data type.

The TOD data type has the following format:

TIME_OF_DAY | time_of_day | TOD | tod#HH:MM:S.S, where HH (0-23) is the hour, MM (0-59) is the minute and S.S (0.0-59.0) is a real number containing seconds.

The NOW system variable is a TOD data type that contains the current system time and can be used to determine when an event takes place. These operators can be used with NOW: EQ, LT, GT, LE, GE, and NE. To read the value of NOW, use the assignment statement or MOVE command to move the value to a variable.

If you create an expression of TOD data types, inputs and outputs must be the data types listed in "Using Date/Time-Based Data Types in Expressions."

TIME

The TIME data type is a member of the ANY group of data types. TIME data types are valid in any instruction or function block that accepts ANY or TIME data types. TIME is a variable that represents a duration of time.

The TIME data type has the following format:

TIME | time | T | t# followed by a sequence of one or more numbers and time unit specifiers. The time unit specifiers, ranges, and examples of their usage are listed below. You can separate the specifiers with the underscore character: t#5m_45s.

Time Specifiers

D or d = Days	(0-1000000)	T#1D2h: 1 day and 2 hours
H or h = Hours	(0-23)	t#20H: 20 hours
M or m = Minutes	(0-59)	t#5m45s: 5 minutes and 45 seconds
S or s = Seconds	(0-59)	t#26S200MS: 26 seconds and 200 milliseconds
MS or ms = Milliseconds	(0-999)	T#45.325ms: 45.325 milliseconds

When a variable of a TIME data type is converted to one of the ANY_NUM data types, the value is converted to a number of seconds. For example, when 3 minutes and 24 milliseconds is converted to a REAL number, the value is 180.024 seconds.

You can use decimal equivalents of days, hours, etc., but only the least significant unit can be fractional. For example, t#1.5d is equivalent to t#1d12h. t#1d1.5h is equivalent to t#1d1h30min. However, t#1.5d6h is not valid.

If you create an expression of TIME data types, inputs and outputs must be the data types listed in "Using Date/Time-Based Data Types in Expressions."

Using Date/Time-Based Data Types in Expressions

If you create an expression of ANY_DATE data types, inputs and outputs must be the data types listed in the following table.

Time-Based Data Types Used in Expressions

Operation	Input1	Input2	Output
Addition	TIME	TIME	TIME
Addition	TIME_OF_DAY	TIME	TIME_OF_DAY
Addition	DATE_AND_ TIME	TIME	DATE_AND_ TIME
Addition	DATE	TIME	DATE
Addition	DATE	TIME_OF_DAY	DATE_AND_ TIME
Subtraction	TIME	TIME	TIME
Subtraction	DATE	TIME	DATE
Subtraction	DATE	DATE	TIME
Subtraction	TIME_OF_DAY	TIME	TIME_OF_DAY
Subtraction	TIME_OF_DAY	TIME_OF_DAY	TIME
Subtraction	DATE_AND_ TIME	TIME	DATE_AND_ TIME
Subtraction	DATE_AND_ TIME	DATE_AND_ TIME	TIME
Subtraction	DATE_AND_ TIME	DATE	TIME_OF_DAY
Subtraction	DATE_AND_ TIME	TIME_OF_DAY	DATE

TMR

The TMR data type is a member of the ANY group of data types. TMR data types are valid in any instruction or function block that accepts an ANY or TMR data type.

The TMR has four system variables, which are identified by the timer name plus an extension:

- Tmr_name.PT contains the preset time value and is a TIME data type. This variable is retentive (retains its value during a power loss). For more information about retentive variables, see "Retentive Variables."

  You can specify the initial value for this variable in these ways:

  Enter a value in the **Symbol Properties** dialog box in the Symbol Manager.

  Use assignment statements (Structured Text program) or MOVE function blocks (RLL program) to assign an initial value to this variable.
- Tmr_name.EN starts/stops the TMR and is a BOOLEAN data type.
- Tmr_name.ET contains the elapsed time of the TMR in seconds and is a TIME data type.
- Tmr_name.Q represents the TMR output and is a BOOLEAN data type.

Operation of the TMR is as follows:

- When tmr_name.EN transitions from FALSE to TRUE, tmr_name.ET is set to zero, tmr_name.Q is set to FALSE, and the timer begins to time.
- When tmr_name.ET equals tmr_name.PT, then tmr_name.EN is set to FALSE and tmr_name.Q is set to TRUE. You can design the program to reset tmr_name.EN earlier than the preset time.
- If tmr_name.EN is held TRUE, by an Assignment statement, for example, tmr_name.Q will be TRUE for the duration of one scan with a cycle period of tmr_name.PT. You can use tmr_name.Q in an IF condition to cause code to execute cyclically. The cycle will drift by approximately one scan.
- If tmr_name.EN is set to FALSE, tmr_name.ET is frozen at its last value and tmr_name.Q remains FALSE.
- The elapsed time tmr_name.ET can be read at any time.

If tmr_name.EN is set back to TRUE on the same scan that the timer expires, then the timer will maintain the excess rollover for the next time interval so that a regular time pulse can be maintained. For example, a one minute timer will generate a pulse event with tmr_name.Q field every minute with no drift due to round off error.

Note Timers evaluate actual time elapsed and are not affected by setting a program or the runtime engine to Paused mode.

ANY

The ANY data type is a generic data type. ANY can assume the type and range of any of the data types that are supported by InControl with these exceptions: FILE, TMR, and User-Defined.

You can use the ANY data type in arrays and in user-defined functions for the return type, local symbols and parameters. You cannot use the ANY data type in user-defined function blocks.

You can use the ANY data type on the left side of an Assignment statement, as shown below:

```
ANY_Vari := INT_Var;
```

You cannot use the ANY data type as part of a complex expression. The code shown below is not valid.

```
ANY_Vari := ANY_Vari * 3.14 + 100;
```

FILE

The FILE data type is a member of the ANY group of data types. FILE is a structure that is designed only for the file control variables used with the RLL and Structured Text file functions.

The FILE data type has several system variables, which are identified by the function control block name (fcb) plus an extension. Three of these variables specify status of a file after it is open: read/write, whether data can be appended, and whether other applications can access the file. Eight variables provide a means of monitoring errors, whether a file is in use, when an operation is completed, etc.

The three input variables are listed in the File Control Input Variable table, below.

File Control Input Variables

Variable	**Description**
fcb.ACCESS [1]	Byte variable specifies read/write status of the file after it opens. FileAccess.ReadWrite = (default) file is open for read/write operations. FileAccess.Read=file is open for read-only operations. FileAccess.Write=file is open for write-only operations.
fcb.APPEND [1]	Boolean variable specifies whether data can be appended to the file after it opens. Only valid when file is open with write status. That is, the ACCESS variable = 0 or 2. TRUE = data will be appended to the file. FALSE = (default) data cannot be appended to the file.

Variable	Description
fcb.SHARE [1]	Byte variable specifies how other applications can access the file after it is open. FileShare.ReadWrite=(default) other applications can access the file for read-write operations. FileShare.Read=other applications can access the file for read-only operations. FileShare.Write=other applications can access the file for write-only operations. FileShare.None=other applications cannot access the file.
1 These variables are read and take effect only when the STL OPENFILE and NEWFILE functions or RLL FOPEN and FNEW function blocks are executed.	

The seven variables that handle file operations are listed in the following table.

File Control Output Variables

Variable	Description
fcb.BUSY	Boolean variable indicates that the file is being accessed. The system sets the File Control Busy variable to TRUE when the file is being accessed by another file function. If you attempt to execute a file type function while this variable is TRUE, an error occurs (error code 15).
fcb.EFLAG	Boolean variable indicates when an error occurs. If an error occurs during a file operation, the system sets the File Error variable to TRUE. This variable is not reset automatically; the program must reset the variable. You can also reset it manually through the Watch window. A file type function cannot execute while this variable is TRUE. The program does not go into Fault mode when an error occurs.
fcb.EOF	Boolean variable indicates that the system encountered an End Of File. The system sets the End Of File variable to TRUE when it encounters the EOF.
fcb.ERR	Integer variable contains the error code if an error occurs. If an error occurs during a file operation, the system writes an error code to the File Error Code integer. The table that follows lists the error codes.
fcb.OPEN	Boolean variable indicates the file has been opened. The system sets the File Open variable to TRUE when the file is open.
fcb.RDN	Boolean variable indicates that a read operation has been completed. The system sets the File Read Done variable to TRUE when the read operation is finished.
fcb.WDN	Boolean variable indicates that a write operation has been completed. The system sets the File Write Done variable to TRUE when the write operation is finished.

STRING

The STRING data type is a member of the ANY group of data types. STRING data types are valid in any instruction or function block that accepts an ANY or STRING data type.

The format for a STRING data type consists of a string of ASCII characters in single quotation marks. Example: ‘This is a valid string.’ The maximum length of a STRING data type is 1024 characters. A string is terminated by the NULL character in those string operations by all STL and RLL functions except for string/array conversions. For more information about these exceptions, see STOBA and BATOS in the "RLL Program Elements" chapter and STRING_TO_ARRAY and ARRAY_TO_STRING in the "Structured Text Language" chapter.

InControl interprets a $ followed by two hexadecimal digits, enclosed in single quotation marks, as the hexadecimal representation of the eight-bit character code. Example: ‘ $41 $42 $43 ’ is interpreted as A B C.

To designate special characters in a string, precede them with the dollar sign, as shown in the following examples:

String Special Characters

Dollar sign = ‘ $$ ’	New line = ‘ $N ’ or ‘ $n ’
Single quote = ‘ $’ ’	Form feed = ‘ $P ’ or ‘ $p ’
Double quote = ‘ $” ’	Carriage return = ‘ $R ’ or ‘ $r ’
Line feed = ‘ $L ’ or ‘ $l ’	Tab = ‘ $T ’ or ‘ $t ’

RTEMODE

The RTEMode data type is termed an enumeration type. That is, the values that you can assign to a symbol of this data type are limited to a set of constant strings. Valid values that you can write to an RTEMode data type are COMPLETE, FAULT, LOADED, PAUSE, PROGRAM, RUN, SCAN, STOP, UNKNOWN.

User-Defined

The User-Defined data type is a member of the ANY group of data types. User-Defined data types are valid in any instruction or function block that accepts a User-Defined data type, or one of the data types in the ANY group.

The User-Defined data type can be either a structure or an enumeration.

- A structure consists of a set of existing data types (INT, WORD, REAL, other user-defined data types, etc.), which are called members. The ANY, TMR and FILE data types and functions and function blocks are not valid members for the user-defined data type. For each structure, you define the members and the data type for each member. The members of a structure do not have to be the same data type. You can use individual members anywhere within a program where the data type for that member is valid.
- An enumeration is a type of structure, the members of which are a set of DINT data types. Use an enumeration when you need to define a group of named constants. When logic that contains the constant is solved, the value of the constant is the initial value, which you assign when you define the constant.

Example of the use of a structure: you can create a user-defined data type called Device_Status, composed of three members called Running, Stopped, and Speed. Running and Stopped are BOOL data types, and Speed is a REAL data type. You can then define a variable, e.g., Motor1, which is a Device_Status data type, and it automatically has the three associated members Motor1.Running, Motor1.Stopped, and Motor1.Speed. In an RLL program, a coil called Motor1.Running controls the startup of Motor1; a coil called Motor1.Stopped stops Motor1; and the speed of Motor1 is determined by the value contained in Motor1.Speed.

If you want Device_Status to have code associated with it, define it as a function block. For more information, see "Function Blocks" in the "Program Organization and Management" chapter.

Example of the use of an enumeration: The seven modes of the runtime engine are represented by members of the enumeration called Mode, of type RTEMode. The members and their initial values are Fault (6), Run (5), Program (4), Scan (3), Pause (2), Stop (1), and Unknown (0). When the runtime engine is running, for example, Mode = Run. If you assigned the value of Mode to an integer variable, the variable takes the value of 5.

Data Type Conversion

When mixed data type operations are encountered during program compilation, the compiler may convert the data type to a larger numeric type. Warnings may be generated when the compiler converts data types. You can avoid some of these warnings by using the conversion functions. An attempt to mix incompatible data types when no valid conversion or promotion exists generates an error message.

Conversion is handled for these data types:

- BYTE
- WORD
- DWORD
- DINT
- INT
- SINT
- REAL
- LREAL
- BOOL (limited in scope)

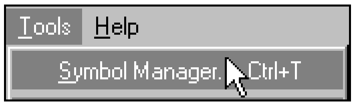
Most of the function blocks accept a variety of input and output data types and convert them as needed. The conversion works on an operation-by-operation basis and may hide intermediate results that are out of range. If this intermediate result is significant, then you need to convert one or both of the operands to larger numeric data types. For example, if you use the RLL ADD function block to add two WORDs, the sum may require a DWORD. Because the result does not fit in a WORD, it is truncated. In this case, changing one or both of the operands to DWORDs yields a different (untruncated) value. It is necessary to convert one or both WORDs to DWORDs before adding them.

Accessing the Symbol Manager

Use the Symbol Manager to create or edit program variables. You can access the Symbol Manager from various points of program development. Some of these are described below. The Symbol Manager fields and buttons are described in the "Symbol Manager Dialog Box" table (page 6-21).

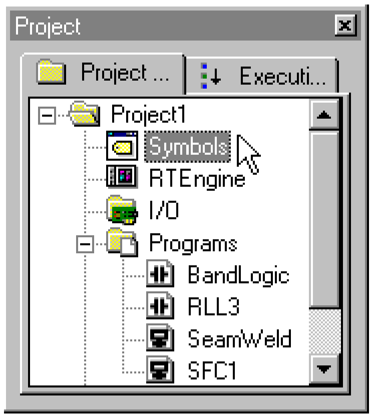
To access the Symbol Manager:

On the Tools menu, click Symbol Manager.



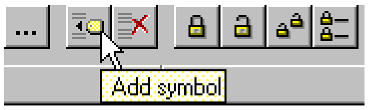
OR

In the Project Window, double-click **Symbols**.



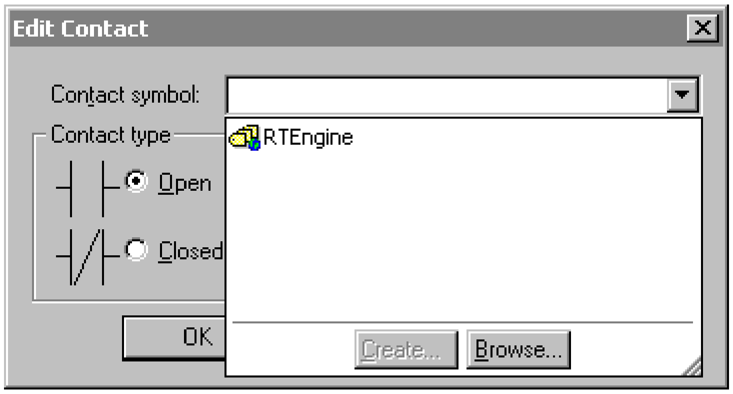
OR

In the Watch Window, click **Add Symbol**.



OR

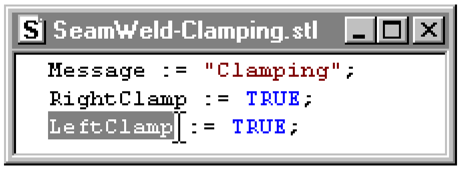
In the **Edit Contact** and **Edit Coil** dialog boxes, and the RLL function blocks, click the Contact Symbol drop-down menu, then click **Browse**.



OR

To access the Symbol Manager:

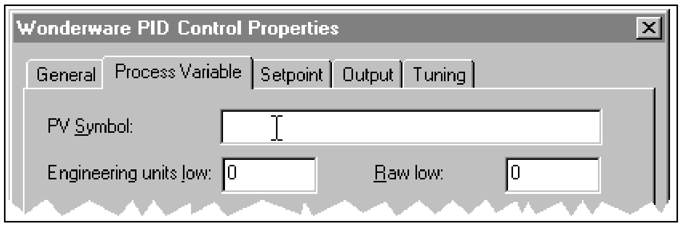
Double-click any variable (or any blank area) in a Structured Text program.



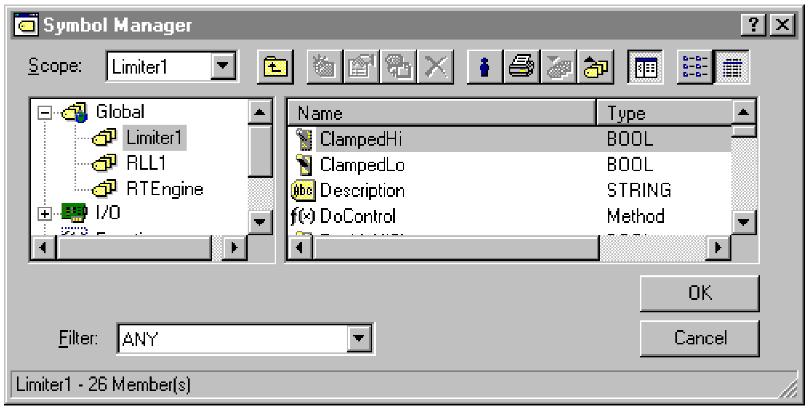
```
Message := "Clamping";
RightClamp := TRUE;
LeftClamp := TRUE;
```

OR

Double-click selected variable fields in the configuration dialog boxes for some FOEs.



The Symbol Manager is shown in the following figure.



Name	Type
ClampedHi	BOOL
ClampedLo	BOOL
Description	STRING
DoControl	Method

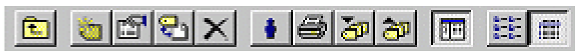
Limiter1 - 26 Member(s)

Symbol Manager Dialog Box

Field	Description
Scope	Click the display tool ▼ to display the scope of variable: global variables, individual (local) program variables, functions, function blocks, macros, runtime engine system variables, or user-defined variable type definitions. You can also choose the scope of variables by clicking the appropriate level in the tree structure.
Name	Displays the names of all variables in the selected scope.
Type	Displays the data type of the variable.
Address	For I/O drivers, specifies the location of the I/O point. For user-defined functions, function blocks, and data types, specifies the order in the structure.
Description	Displays the variable descriptions.
Filter	Use the filter to display only selected data types.

Using the Symbol Manager Toolbar

The Symbol Manager toolbar displays the tools used to create and handle symbols.



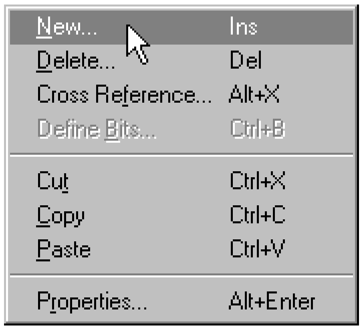
Symbol Manager Toolbar

Option/Button	Description
	Click **Up One Level** to move up one level in the scoping hierarchy.
	Click **New** to add a variable to the Symbol Manager.
	Click **Properties** to edit an existing variable.
	Click **Cross Reference** to display variables, where and how often they are used in the program.
	Click **Delete** to remove a variable from the Symbol Manager.
	Click **Edit User Types** to create/edit a user-defined data type.
	Click **Print Symbols** to print a list of the variables in the Symbol Manager. You can also choose to print cross-references and variables that are not referenced.
	Click **Import** to read an ASCII file (comma-separated variable format) of variables into the InControl database.
	Click **Export** to create an ASCII file (comma-separated variable format) of the InControl variables.
	Click **Show/Hide Tree** to display or hide the symbol tree structure.
	Click **List View** to display a variable listing only.
	Click **Detailed View** to display a variable's name, data type, address, and description.

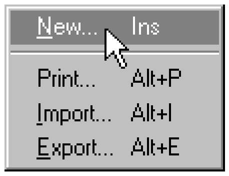
Editing Tips - Context Menus

During a symbol editing session, you can right-click for a fast display of the editing options for the Symbol Manager.

- With a symbol selected, right-click to display the following menu:



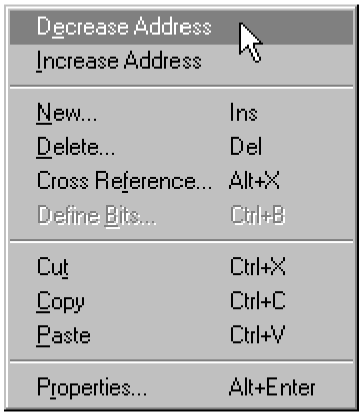
- With no symbols selected, right-click to display some of the options in the Symbol Manager toolbar:



Editing Tips - Changing Member Order

You can change the order of the members in a user-defined data type or the parameters of a user-defined function or function block.

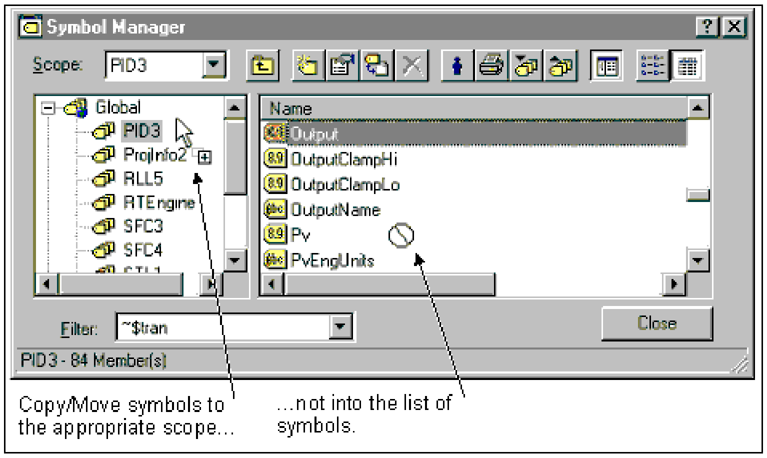
- With the object selected, right-click and use the **Decrease / Increase Address** options.



Editing Tips - Copy / Paste / Move Symbols

You can copy/paste symbols as described below.

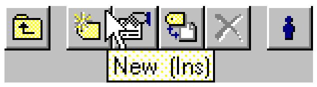
- Between the local and global scope within a program. You can also drag symbols between scopes.
- Between the Symbol Manager and other applications.
- Between the Symbol Managers of two instances of InControl. When you select a symbol, drop it on the appropriate scope (Global, program name, or User Type Definition), not in the symbol list. See the following figure.



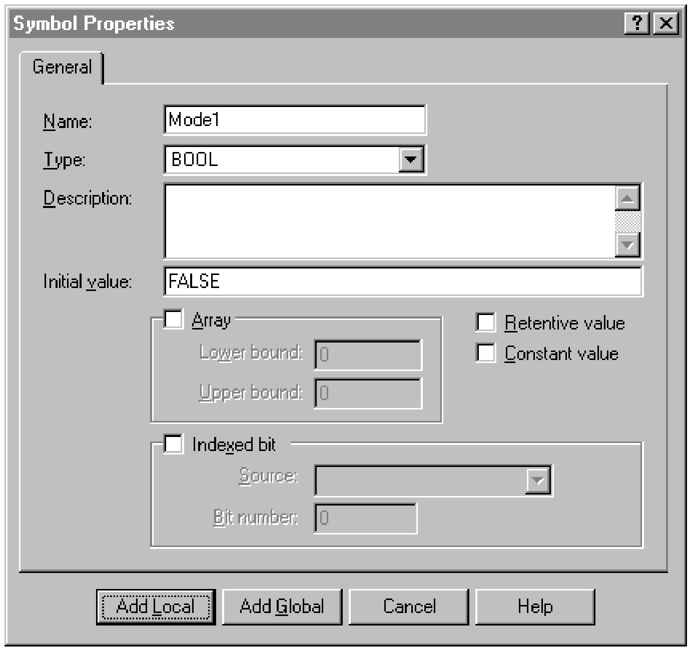
Creating a Variable

To create a program variable:

1. Access the Symbol Manager.
2. Select the appropriate scope, for example, global, local (an individual program), function block, etc.
3. Click **New** on the Symbol Manager toolbar.



The **Symbol Properties** dialog box appears displaying the same properties as the preceding variable in the list.



Name: Mode1
Type: BOOL
Description:
Initial value: FALSE
Array: ☐
Lower bound: 0
Upper bound: 0
Retentive value: ☐
Constant value: ☐
Indexed bit: ☐
Source:
Bit number: 0

Creating a Variable - Symbol Properties Dialog Box

4. Enter the name of the variable into the **Name** field. Use only alphanumeric characters and the underscore character.

   Local variables must be unique. That is, two variables with the same name cannot be in the same program. In addition, a local variable cannot have the same name as a global variable.
5. Select the data type in the **Type** field.
6. Enter the optional description into the **Description** field.
7. If the variable requires an initial value, enter the value into the **Initial Value** field.
8. Check the **Constant** checkbox if you want the value for this variable to remain constant. Be sure to enter the value in the **Initial Value** field.

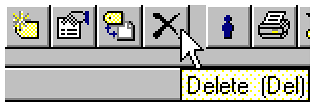
9. Click the **Retentive Value** checkbox if the variable is to retain its value in the event of a power loss.

   InControl periodically saves retentive and forced variables to the hard disk. The default frequency of zero disables this feature. This interval is configurable and you can set it in the Runtime Engine Properties dialog box, described in "Setting Scan Times" of the "InControl System Administration" chapter.

10. Click **Add Local** or **Add Global**. The new variable appears in the **Symbol List** field.

To remove an existing variable:

1. Click the name of the variable.
2. Click **Delete** on the Symbol Manager Toolbar.



You can also press **Del** on the keyboard.

Creating an Array of Variables

To create an array of variables:

1. Access the Symbol Manager and follow the steps described in "Creating a Variable" for creating a new variable.
2. Check the **Array** checkbox. This selects for an array of variables and the **Lower Bound** and **Upper Bound** fields become active.
3. Enter lower and upper values for the array into the **Lower Bound** and **Upper Bound** fields.
4. Click **Add Local** or **Add Global**.

Referencing Arrays

Use literal values or expressions in brackets to reference the elements of an array. In the following example, an element is referenced with a literal value.

```
Array_of_Values[0] := 42;
```

In the following example, an element is referenced with an expression.

```
Int1 := Array_of_Values[i+1];
```

Note The InControl compiler generates an out-of-range error if you use a literal value for an array index and it is not within the specified bounds. If you use an expression to define the index, and the expression resolves to a value that is out of range at runtime, the program enters the Fault mode.

Assigning a Name to a Bit in a Variable

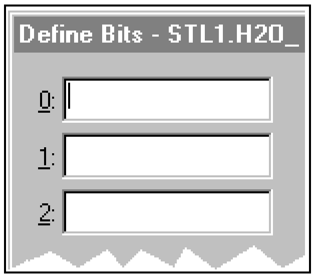
The Indexed Bit feature allows you to reference a specific bit within a BYTE, WORD, or DWORD variable. The scope must be as follows:

- If the source variable is global, the bit can be either local or global.
- If the source variable is local (scoped to a program, or to a runtime engine variable, for example) the bit must have the same scope.
- If the source variable is a member in a user-defined data type, the bit must be a member in the same user-defined data type.

You can select the source variable and assign names to the bits, or define the bit and then specify its source variable.

To assign names to bits in a selected variable:

1. Access the Symbol Manager.
2. Right-click the source variable (BYTE, WORD, or DWORD data type) and then click **Define Bits**. The **Define Bits** dialog box appears.



Define Bits Dialog Box

3. Enter the name for the bit(s).

To define a bit and then specify its source variable:

1. Access the Symbol Manager and follow the steps described in "Creating a Variable" for creating a new variable.
2. Check the **Indexed Bit** checkbox.
3. Enter the name of the source variable in the **Source** field.
4. Enter the bit number in the **Bit #** field. Valid values are 0-7 for a BYTE, 0-15 for WORDs, and 0-31 for DWORDs.

   If the source is an array, enter the bit number based on its position in the array. For example, to index the last bit in an array of four bytes, enter 31 for the bit number.
5. Click **Add Local** or **Add Global**.

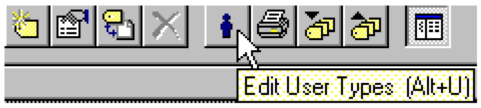
Creating a User-Defined Data Type

You can custom-design a data type for specific applications. First, you create a user type definition and then add variables to your program that are based on the definition. The user type definition can be either a structure or an enumeration, as described in "User-Defined."

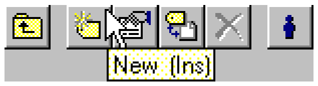
Custom-Designing a Data Type

To create a user type definition:

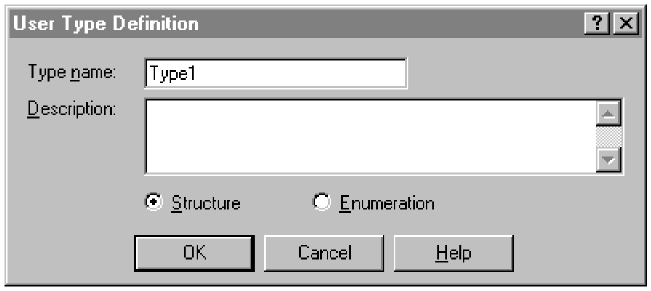
1. Access the Symbol Manager.
2. Click **Edit User Types** on the Symbol Manager toolbar.



3. Click **New** on the Symbol Manager toolbar.



The **User Type Definition** dialog box appears.



Type name: Type1

Description:

[✓ Structure] Enumeration

User Type Definition Dialog Box

4. Enter a name in the **Type Name** field, and a description (optional). Click **Structure** or **Enumeration** and then click **OK**.
5. To begin adding members to the structure or enumeration, click **New** on the Symbol Manager toolbar. The **Symbol Properties** dialog box appears.
6. Enter the appropriate information, as described in "Creating a Variable." For enumerations, the data type must be a DINT.
7. Click **Add Member**. Then repeat steps 5-7 to add additional members to the structure.

You can change the order of the members in a user-defined data type. With the data type selected, right-click and use the **Decrease / Increase Address** options. This allows you to modify an existing user-defined data type instead of defining a new one.

Note If you make changes to the user type definition, such as adding or removing a member, all instances of that definition are automatically updated. However, if you specify an initial value for a member and later change the initial value, the initial values of any variables that are based on the originally defined data type are not updated. Such changes, including a change to the address order of the members, require a full reload of the project before you can run it.

Using the User-Defined Data Type

The new user type definition will appear as one of the choices in the **Type** field of the **Symbol Properties** dialog box, along with the other data types. When you define a new variable in the Symbol Manager, you can choose the new user type definition as the data type for the variable.

The following examples demonstrate various ways to reference user-defined types.

Referencing an individual member:

```
User_Type1.Int1 := 42;
```

Referencing an array of user-defined types:

```
User_Type2[6].Int1 := 11;
```

Referencing an array of user-defined data types with a member that is an array:

```
User_Type3[9].Int2[3] := 77;
```

Note The InControl compiler generates an out-of-range error if you use a literal value for an array index and it is not within the specified bounds. If you use an expression to define the index, and the expression resolves to a value that is out of range at runtime, the program enters the Fault mode.

Printing Information for Variables

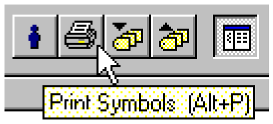
You can print the following types of reports from the Symbol Manager.

- Symbol details—Data type, description, and initial value for each variable.
- Program references—All variables that are referenced by each program.
- Cross references—All programs that reference each variable.
- Unreferenced—Variables that have been defined but are not referenced.

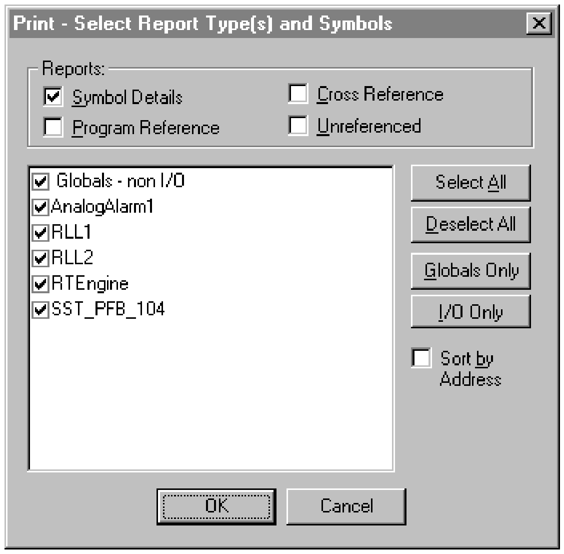
The InControl compiler generates the information that is printed. Therefore, you must validate the project before you print a report.

To print information for variables:

1. Validate the project.
2. Access the Symbol Manager.
3. Click **Print Symbols** on the Symbol Manager toolbar.



The **Print Select Report Type** dialog box appears.



Print Select Report Type dialog box

4. Select the type of type of report and the variables (global, local, I/O, etc.).

Note If you print more than one type of report, you are prompted to select the printer for each report.

System Variables - General

InControl provides system variables that can be used to monitor and control various functions of the system. The following table lists the general system variables.

General System Variables

Variable	Description
Mode	A Mode variable (INT) is created for each program and also for each I/O configuration. These variables can have the following values: UNKNOWN (0) Unloaded from the runtime engine. STOP (1) Program (I/O) is stopped. PAUSE (2) Program (I/O) is paused. SCAN (3) Program (I/O) is in single scan mode. PROGRAM (4) Program (I/O) is being loaded to runtime engine. RUN (5). Program (I/O) is running. FAULT (6) Program (I/O) is in error. To clear a fault, see "Clearing Fault Mode and Error Conditions" in the "InControl System Administration" chapter. COMPLETE (7) SFC program has finished execution. Other program types can be set to the Complete mode. Functionally, the Complete and Paused modes are equivalent. LOADED (8) **For programs**: program is loaded to the runtime engine and is ready to run when called by another program. Applicable to functions, function blocks, and FOEs that require function calls to a method in order to run. Note that the mode of these POUs does not change to Run, even after they are called. **For I/O configurations**: configuration has been downloaded to a remote node. Automatic configuration can be done on that remote node while the configuration is in the loaded mode.
NOW	Contains the current system time. For more information, see "TOD."
TODAY	Contains the current system date. For more information, see "DATE."
T [1]	Contains the elapsed execution time of an SFC step.
X [1]	Contains the active/inactive status of an SFC step.
DN [2]	For an SFC, indicates when an SFC is finished executing. For a Step, indicates when the code within a Step is finished executing.
1 For more information, see the "SFC Program Elements" chapter.	

System Variables - Runtime Engine

InControl provides system variables for the runtime engine that can be used to monitor and control the runtime engine. The following table lists the runtime engine system variables.

Note The general system variables generated by the runtime engine do not appear on a newly installed system or when you stop the runtime engine (click **Stop** on the **Runtime** menu). In these situations, you cannot add them to the Watch window and SuiteLink/DDE clients cannot read them.

Runtime Engine System Variables

Variable	Description
RTEngine.DivideZero	Boolean value. TRUE indicates a division by zero; set by the runtime engine (RTE).
RTEngine.Error	Boolean value. TRUE indicates an error condition has occurred in the runtime engine. Check the Wonderware Logger for more information.
RTEngine.ExecAvg	TIME value. Contains the average execution time for the program logic that is being executed.
RTEngine.ExecLast	TIME value (read only). Contains the last execution time for the program logic that is being executed.
RTEngine.ExecMax	TIME value. Contains the maximum execution time for the program logic that is being executed.
RTEngine.FirstScan	Boolean value (read only). TRUE indicates occurrence of first program logic scan.
RTEngine.FirstScanOnAutoStart	Boolean value (read only). TRUE indicates occurrence of first program logic scan after an automatic start following a system reboot.
RTEngine.IOAvg	TIME value. Contains the average I/O scan time.
RTEngine.IOLast	TIME value (read only). Contains the last I/O scan time.
RTEngine.IOMax	TIME value. Contains the maximum I/O scan time.
RTEngine.Mode [1]	RTEMode value. Indicates current mode of the runtime engine: UNKNOWN (0) All programs are unloaded from runtime engine. STOP (1) Programs in a project are stopped. PAUSE (2) Programs in a project are paused. SCAN (3) Project is in single scan mode. PROGRAM (4) Project being loaded to runtime engine. RUN (5) At least one program in a project is running. FAULT (6). Runtime engine cannot run project.

Variable	Description
RTEngine.NodeName	STRING value that contains the name of the node where the runtime engine is running.
RTEngine.PowerFail	Boolean value indicates a power failure when TRUE. A UPS configuration is required.
RTEngine.ProjectName	STRING value that contains the name of the project currently loaded in the runtime engine.
RTEngine.RelativeTime	TIME value (read only) that contains the length of time that the runtime engine has been running since the system was booted. This value is independent of the system clock and can be used in a program to calculate timed intervals.
RTEngine.ScanAvg	TIME value contains the average scan.
RTEngine.ScanLast [2]	TIME value (read only) contains duration of the last scan.
RTEngine.ScanMax [2]	TIME value contains the maximum scan time.
RTEngine.ScanOverrun [3]	Boolean value indicates a scan overrun when TRUE.
RTEngine.ScanTime	LREAL value contains the current user-assigned scan time setting in milliseconds. Values written to this variable will change the runtime engine scan time.
RTEngine.TimeStamp	DT value that contains the timestamp of the project currently loaded in the runtime engine.
RTEngine.Version	STRING value that contains the version of the runtime engine.

1 The Watch window and external programs, such as InTouch, or a Visual Basic application, can write to Mode, and the new value is displayed for the variable. InControl executes any changes written to the variable. You can write or force only the values 2, 3, 5, or 6 to the RTEngine.Mode variable.

2 The RTEngine.ScanLast and RTEngine.ScanMax variables may have incorrect values at very fast scan times. The time monitoring utilities execute at a lower priority than the runtime engine. Occasionally more than one scan may occur without being timed. Check the RTEngine.ScanOverrun bit to see if scans are taking longer than expected to complete. For more information, see "Adjusting the Scan Time" in the "InControl System Administration" chapter.

3 For a discussion of the conditions that can result in a scan overrun, see "Setting Scan Times" in the "InControl System Administration" chapter.

Note The RTEngine.RelativeTime variable can be used for timing events, as shown in this example:

```
IF (B1) THEN
SavedTime:=RTEngine.RelativeTime;
ENDIF;
IF (B2) THEN
TimeItTook:=RTEngine.RelativeTime-SavedTime;
ENDIF;
```

Transferring Symbol Databases

InControl supports the exchange of symbols between InControl projects and between InTouch and InControl. The export/import utility can write and read an ASCII text file that has a comma-separated format (CSV). You can easily edit a file of symbol data using an ASCII text editor or a spreadsheet, such as Excel.

The export utility gives you the following options for exporting symbol information.

- InControl format
- InControl cross reference
- InTouch format
- InTouch super tags

Symbol Exchange Between InControl Projects

When you transfer symbols between InControl projects, consider the following points.

- Use the InControl file format when you export symbols. When you import symbols, this format is used automatically.
- InControl symbols are created by a number of different kinds of objects. Some of these objects do not permit importation of symbols, for example, ActiveX, FOEs, and I/O objects.
- You can load new local symbols into a program, but the program must already exist. For example, to load local symbols into a program named RLL1, you must create a program named RLL1 before importing the symbols. Note that the same is true for symbols used in functions, function blocks, and macros; these POUs must already exist before you can import their symbols. Global symbols can be imported without this limitation.

Symbol Cross-Reference Reports

You can create a report of symbol cross references that has a comma-separated format. You can open the report with any ASCII text editor, such as Notepad, or with a spreadsheet, such as Excel.

Symbol Exchange Between InControl and InTouch

When you transfer symbols between InControl and InTouch, consider the following points.

- Use the InTouch file format or the InTouch Super Tag file format when you export symbols to InTouch. Choose the Super Tag format to create InTouch super tags from logical groups of InControl symbols. The following groups are examples of InControl symbols that can be exported as super tags:

  Elements of an array.

  Parameters of timers or counters.

  Parameters of any function or function block.

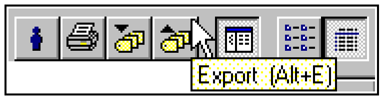
  Runtime engine system variables.

- Symbols exported from InTouch can only be imported as global symbols to InControl; and only those marked as being items of the runtime engine are imported.
- When importing symbols to InTouch, the symbol names are translated to the InTouch-compatible format in the same way the wizards do in InControl. All periods and brackets are converted to underscores. That is, GArray[1] becomes GArray_1_ and RTEngine.ScanMax becomes RTEngine_ScanMax.

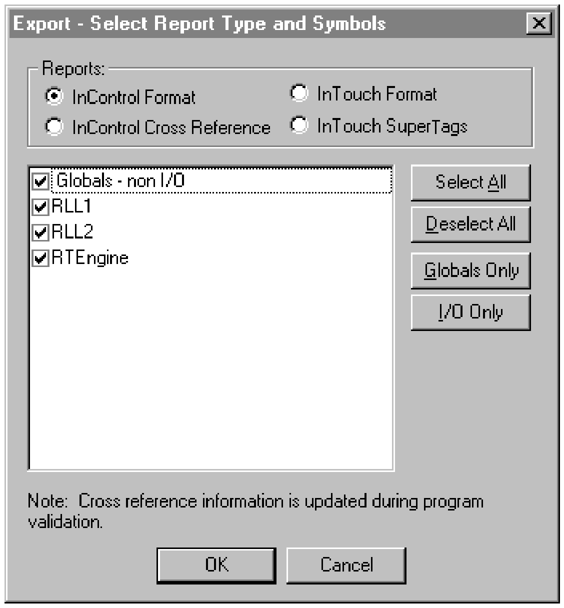
Importing/Exporting Symbols

To export symbols:

1. Access the Symbol Manager and click **Export** on the Symbol Manager toolbar.

   The **Export - Select Report Types** dialog box appears.

   Export - Select Report Type and Symbols

   Reports:
   - (●) InControl Format
   - ( ) InControl Cross Reference
   - ( ) InTouch Format
   - ( ) InTouch SuperTags

   - [✓] Globals - non I/O
   - [✓] RLL1
   - [✓] RLL2
   - [✓] RTEngine

   Note: Cross reference information is updated during program validation.

   Export -Select Report Types dialog box

2. Select the type of type of report and the variables (global, local, I/O, etc.).

   To export to another InControl project, click **InControl Format**.

   To export to an InTouch project, click **InTouch Format** or **InTouch SuperTags**.

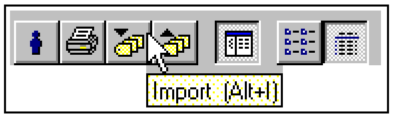
   To generate a file of symbol cross references, click **InControl Cross Reference**.

3. Click **OK**. The **Save As** dialog box appears.

4. Enter a file name and click **Save**. The new file in the CSV format is created.

To import symbols:

1. Access the Symbol Manager and click **Import** on the Symbol Manager toolbar.



2. The **Open** dialog box appears.
3. Select the symbol database file and click **Open**. The symbols are imported and displayed in the Symbol Manager.

InControl CSV File Format

The format of the InControl CSV file is shown in the figure InControl File CSV Format (page 6-39).

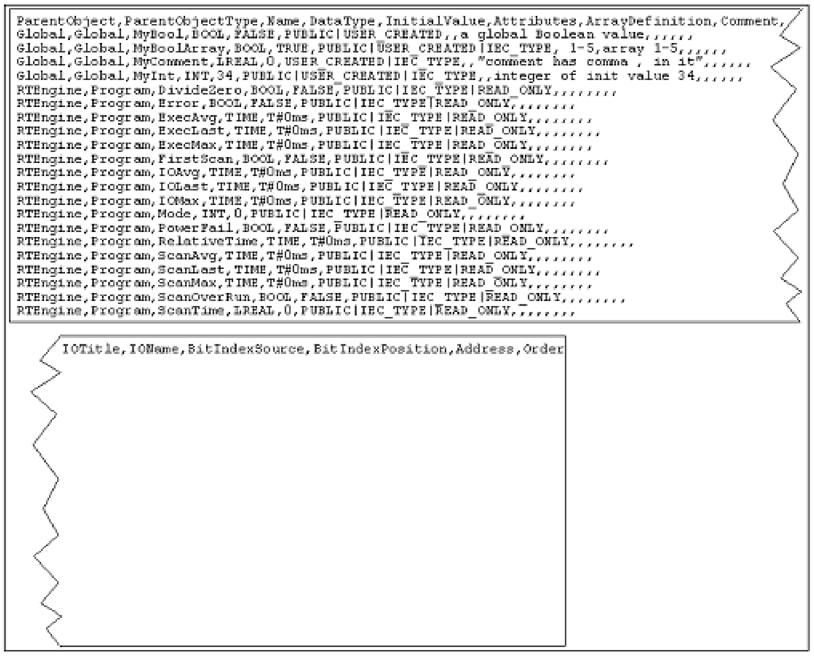
The labels used in the CSV format are listed below.

- ParentObject — name of the object that created the symbol. If global, this is Global.
- ParentObjectType — Global, Program, ActiveX, or IO.
- Name - name of the symbol.
- DataType - BOOL, INT, DINT, BYTE, WORD, DWORD, REAL, LREAL, STRING, TIME, DATE, TOD, or user-defined type.
- InitialValue — initial value of the symbol.
- Attributes — attributes of the symbol, separated by the | character. These are necessary to maintain internal relationships within InControl, and normally should not be edited. If you are creating new symbols, it is recommended that you copy them from an existing symbol with the required characteristics and then make the necessary changes.
- ArrayDefinition — if the symbol is an array, this field contains LowerBound-UpperBound values of the array. If the field is empty, the tag is not an array.
- Comment — this field contains a user-defined description of the symbol.
- IOTitle — if the symbol is an IO point, this field contains the string output by the driver that describes what the symbol represents, for example, PROFIBUS-DP board.Module.Port).
- IOName — if the symbol is an IO point, this field contains the string output by the driver that contains the names corresponding to IOTitle, for example, SMSProfi.OutPutMod.OutPortMs. If the symbol is a bit index into another symbol, this field contains the source symbol name and the bit offset, for example, GWord.0, GWord.1, etc.
- BitIndexSource — if the symbol is a bit index into another symbol, this field contains the name of the source symbol.
- BitIndexPosition — if the symbol is a bit index into another symbol, this field contains the bit position within the source.

- Address — address of the symbol. Most symbols do not have an address.
- Order — contains the order (hexadecimal) of a member within a user-defined data type structure.

The format of the InControl CSV file is shown in the following figure.

InControl File CSV Format



```
ParentObject,ParentObjectType,Name,DataType,InitialValue,Attributes,ArrayDefinition,Comment,
Global,Global,MyBool,BOOL,FALSE,PUBLIC|USER_CREATED,,a global Boolean value,,,,,,
Global,Global,MyBoolArray,BOOL,TRUE,PUBLIC|USER_CREATED|IEC_TYPE, 1-5,array 1-5,,,,,,
Global,Global,MyComment,LREAL,0,USER_CREATED|IEC_TYPE,,"comment has comma , in it",,,,,,
Global,Global,MyInt,INT,34,PUBLIC|USER_CREATED|IEC_TYPE,,integer of init value 34,,,,,,
RTEngine,Program,DivideZero,BOOL,FALSE,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,Error,BOOL,FALSE,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,ExecAvg,TIME,T#0ms,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,ExecLast,TIME,T#0ms,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,ExecMax,TIME,T#0ms,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,FirstScan,BOOL,FALSE,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,IOAvg,TIME,T#0ms,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,IOLast,TIME,T#0ms,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,IOMax,TIME,T#0ms,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,Mode,INT,0,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,PowerFail,BOOL,FALSE,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,RelativeTime,TIME,T#0ms,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,ScanAvg,TIME,T#0ms,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,ScanLast,TIME,T#0ms,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,ScanMax,TIME,T#0ms,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,ScanOverRun,BOOL,FALSE,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
RTEngine,Program,ScanTime,LREAL,0,PUBLIC|IEC_TYPE|READ_ONLY,,,,,,,,
```

```
IOTitle,IOName,BitIndexSource,BitIndexPosition,Address,Order
```

Editing Symbol Files

You can edit symbol definitions with a spreadsheet, such as Excel, or an ASCII text editor and then re-import them into InControl. InControl automatically merges any changes with the existing symbol definitions. It is recommended that you not make any changes to the attributes.

In general, if a string contains a comma, the string must be enclosed in double quote marks ("this comment has a comma, within it").

The order of the records within a CSV file has these constraints for user-defined symbols and indexed bit symbols:

User-defined symbols

- The user-defined type must be first.
- The members for the user-defined type appear second.
- Individual symbols of a user-defined type appear last.

Indexed-bit symbols

- The source symbol must be first.
- The indexed bits must follow the source symbol.

C H A P T E R 7

Using the Factory Object Editor

This chapter introduces the InControl Factory Object editor and tells how to use it to add ActiveX controls to an InControl project.

Contents

Defining a Factory Object

InControl is compatible with the ActiveX Server specification. The InControl Factory Object (FOE) editor is an ActiveX container, which enables you to use ActiveX controls within an InControl project.

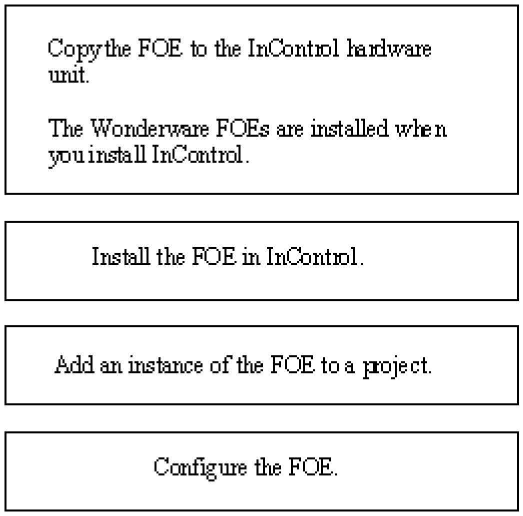
An ActiveX control must be installed within InControl before you can configure and run it. After installation, it is referred to as an InControl factory object (FOE). Like other InControl programs, an FOE can run independently. You can also call it for execution from another program.

The current release of InControl includes several FOEs. The following FOEs are described in this manual:

- Use the PID InControl FOE to handle PID loop functions.For information about configuring the PID FOE, see the *InControl PID and Analog Alarm Reference Manual*.
- Use the Analog Alarm InControl FOE to monitor an analog input signal for alarm conditions. For information about configuring the Analog Alarm FOE, see the *InControl PID and Analog Alarm Reference Manual*.

For information about additional Wonderware Factory Objects, contact your distributor.

To install an FOE in InControl, follow this general procedure.


Copy the FOE to the InControl hardware unit.
The Wonderware FOEs are installed when you install InControl.
Install the FOE in InControl.
Add an instance of the FOE to a project.
Configure the FOE.

Note that InControl FOEs always check for the proper Wonderware licensing information before they are executed. Typically, ActiveX controls are licensed and do not load into InControl if the license is missing.

One or more ActiveX controls may be present on your hardware unit. Note that not all of these are appropriate for running in an InControl project. It is highly recommended that you test third-party objects before using them in a factory process.

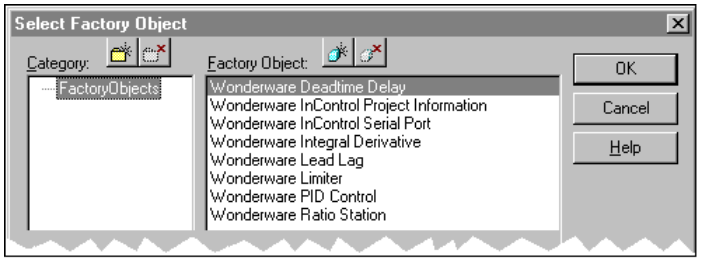
Installing ActiveX Controls

You must install the ActiveX control before you can add it to the Project Window. The Wonderware FOEs are installed automatically when you install InControl.

To install an ActiveX control:

1. Copy the ActiveX control to the InControl hardware unit.
2. On the **File** menu, click **New**.

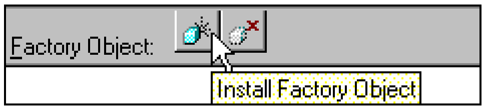
   The **New** dialog box, which lists program types supported by InControl, appears.
3. Select **Factory Object** and click **OK**. The **Select Factory Object** dialog box appears. Note the Wonderware FOEs, which are already installed.



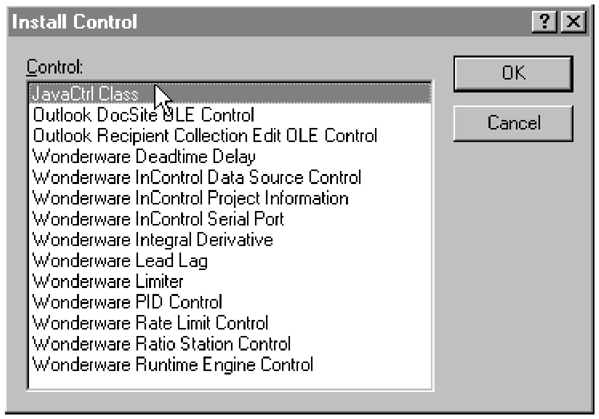
Wonderware Deadtime Delay
Wonderware InControl Project Information
Wonderware InControl Serial Port
Wonderware Integral Derivative
Wonderware Lead Lag
Wonderware Limiter
Wonderware PID Control
Wonderware Ratio Station

Select Factory Object Dialog Box

4. Optional. If you want to group FOEs in categories, create the category (See "Organizing FOEs") and click the category name before installing.
5. Click **Install Factory Object** on the toolbar.



The **Install Control** dialog box appears.



JavaCtrl Class
Outlook DocSite OLE Control
Outlook Recipient Collection Edit OLE Control
Wonderware Deadtime Delay
Wonderware InControl Data Source Control
Wonderware InControl Project Information
Wonderware InControl Serial Port
Wonderware Integral Derivative
Wonderware Lead Lag
Wonderware Limiter
Wonderware PID Control
Wonderware Rate Limit Control
Wonderware Ratio Station Control
Wonderware Runtime Engine Control

Install Control Dialog Box

6. Select the ActiveX control and click **OK**. The **Select Control Type** dialog box appears.

   For information about choosing control types, see "Using Third-Party FOEs."

   If the ActiveX control is on your system but does not appear in the list, you may need to make a change in the system registry. See "Changing System Registry Keys" in the "InControl System Administration" chapter.

7. Select the type of control and click **OK**. The object is installed in InControl and appears in the **Factory Object** list.

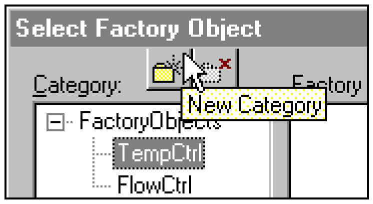
If you click **OK** to close the dialog box, you are prompted to enter a file name and save. This adds the FOE to the project, also described in "Adding FOEs to a Project." If you want to install additional FOEs, repeat steps 4-6 before closing the dialog box.

Organizing FOEs

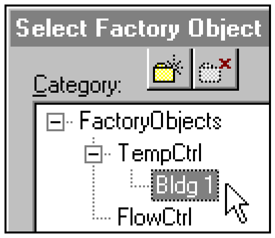
You can organize FOEs under one or more categories and you can uninstall FOEs that you do not need.

To create a new category:

1. Click the category under which you want to create a new one, and click **New Category** on the toolbar.

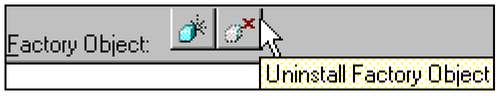


2. When the **New Category** dialog box appears, enter a name and click **OK**. The new category appears in the Category field.



To uninstall an FOE:

1. Click the FOE name.
2. Click **Uninstall Factory Object** on the toolbar.



Adding FOEs to a Project

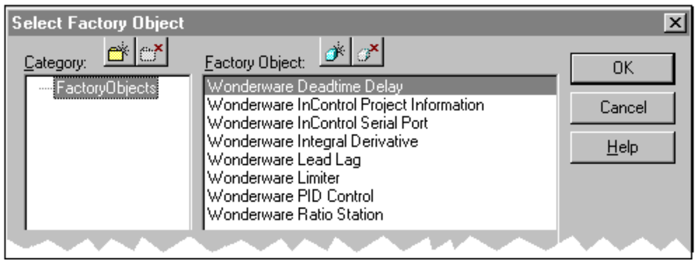
After installing an ActiveX control as an InControl FOE, you can add one or more instances of it to a project.

To add an FOE to a project:

1. On the **File** menu click **New**.

   The **New** dialog box, which lists program types supported by InControl, appears.

2. Select **Factory Object** and click **OK**. The **Select Factory Object** dialog box appears.



Wonderware Deadtime Delay
Wonderware InControl Project Information
Wonderware InControl Serial Port
Wonderware Integral Derivative
Wonderware Lead Lag
Wonderware Limiter
Wonderware PID Control
Wonderware Ratio Station

Adding FOES – Select Factory Object Dialog Box

3. Select the FOE and click **OK**.
4. When the **Save As** dialog box appears, enter a name (up to 31 characters) and click **Save**. The FOE is added to the project.

Configuring Factory Objects

After adding an FOE to a project, you can open it for configuration.

To open and edit an existing FOE:

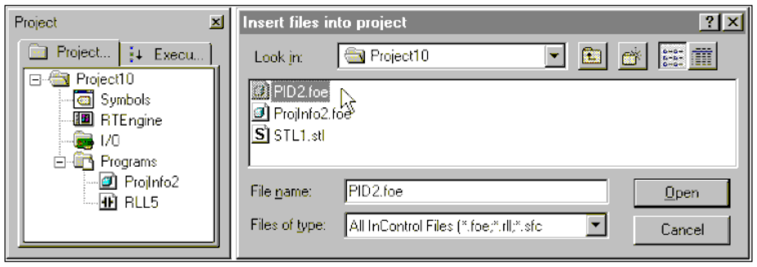
1. If the Project window is not open, click **Project** in the **View** menu to open it.
2. Double-click the name of the FOE. The FOE window opens.
3. To configure the FOE, double-click inside the FOE window.

   The configuration dialog boxes appear. For information about configuring a third-party FOE, refer to the specific documentation for that object. The PID and Analog Alarm FOEs are described in the *InControl PID and Analog Alarm Reference Manual*."

For information about configuring the Serial Port Interface FOE, see the *Wonderware InControl Serial Port Version 2 User's Guide*.

You can also click **Open** in the **File** menu to open an existing program for editing. When the **Open** dialog box appears, select the program to open. If a program is not part of the current project, you can add it.

You can click **Files into Project** in the **Insert** menu to add any POU (program, function, function block, etc.) to a project. In the following figure, the program PID2, shown in the **Insert Files into Project** dialog box, is selected and can be added to Project10.



Look in: Project10

File name: PID2.foe

Files of type: All InControl Files (*.foe;*.rll;*.sfc

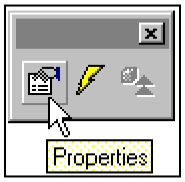
Adding a POU to a project.

Using the Tool and Menu Bars

To configure an FOE, edit its properties. You can open the dialog boxes for the properties by several methods.

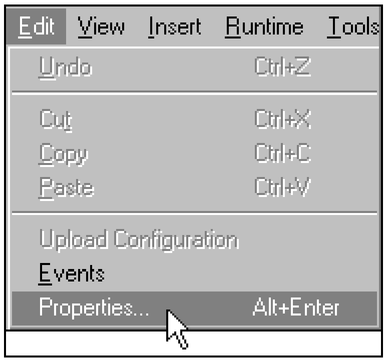
To display the dialog boxes for the properties:

Click the FOE and then click **Properties** on the Factory Object toolbar.



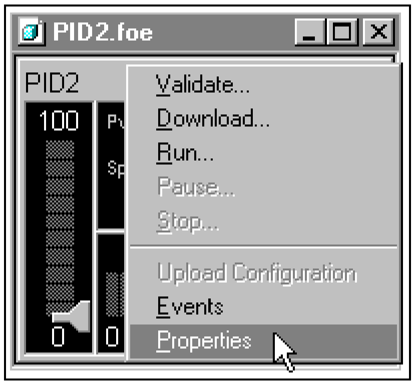
OR

Click the FOE and then click **Properties** on the Edit menu.



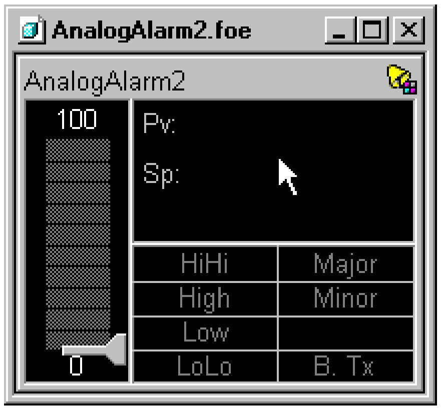
OR

Right-click the FOE and then click **Properties**.



OR

Double-click the FOE. Note that if the FOE is running and you are viewing its animation, you cannot access the properties by double-clicking the FOE. Use one of the other methods to access the properties.



Note You can right-click the FOE to validate, download, run, stop, pause, and upload the FOE.

Running and Controlling FOEs

Some Wonderware FOEs, such as the PID FOE, are designed to run automatically. They have a method that is called once every scan by the runtime engine. For the PID FOE, this method is the DoControl method. When loaded to the runtime engine, the mode for these FOEs is indicated as Run, Pause, Stop, etc.

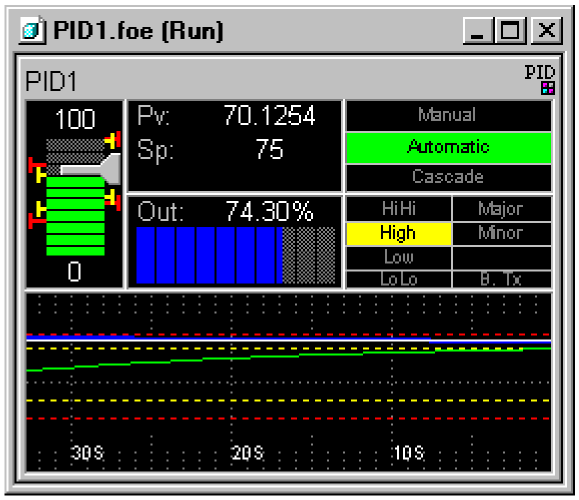
Some FOEs do not have a method designed to run automatically every scan. For these FOEs, you need to include code in an SFC or Structured Text program to call a method for the FOE to execute. When downloaded to the runtime engine, the mode for these FOEs is indicated as Loaded, which is displayed in the Project window. These FOEs are ready to execute methods and set properties when called for execution.

You must also enter code in an SFC or Structured Text program to call one of the FOE methods when you need to interact with the FOE. For example, to set the Wonderware PID FOE named TempControl to automatic mode, the following function call executes the method named Auto:

```
TempControl.Auto();
```

Runtime Animation

Some Wonderware FOEs appear in an animation mode at runtime. If you display one of these FOEs in the Development environment, selected parameters are updated when they change, as illustrated by the Wonderware PID FOE shown below.



PID FOE at Runtime.

To enable runtime animation, on the **View** menu, click **Runtime Highlighting**.

Uploading Parameters

Some FOEs allow you to upload parameters from the runtime engine to the Development environment. This can be useful when you want to determine empirically certain values, and then upload these values to the property sheets for the FOE. Refer to the documentation for the individual FOEs for the specific symbols that are uploaded.

For more information about the behavior of variables at runtime, see the "Defining Variables" chapter.

Using Third-Party FOEs

Some third-party FOEs may not be designed for real-time factory control. InControl FOEs need to operate quickly and not cause memory loss over extended periods of time. If you intend to use the FOE at runtime, you need to verify that it meets the following guidelines.

- Do not use code that makes function calls requiring human interaction, such as a method that displays a dialog box.
- Do not call a method that requires a relatively long period of time to execute, such as sounding the hardware unit speaker causing a message beep.
- Verify that memory is allocated and deallocated correctly for strings that are passed as parameters to methods or returned as values from methods.
- It is highly recommended that you test third-party FOEs before using them in a factory process.
- It is recommended that the threading mode for FOEs be set to "Both." FOEs with threading models that are not set to "Both" and that are used at runtime, can cause a serious degradation in performance. Accessing the data values or methods of the FOE may be up to 20 times slower.

For information about changing the threading model of an FOE, see "Changing FOE Registry Setting" in the "InControl System Administration" chapter.

Note Do not change the threading model from "Apartment" to "Both" for ActiveX Controls built using Visual Basic 5.0 or Visual Basic 6.0.

Event Handling by Factory Objects

Many factory objects have the capability of triggering events. You can map an event to a user-defined function and the function runs when the event is triggered.

Typically, the event takes place synchronously with the runtime engine scan. For example, code in a POU can be written to trigger the event. However, an event can be asynchronous. For example an operator could write a change to a variable in the Watch window and thereby trigger an asynchronous event.

Mapping Functions to Events

The types of events that are supported by a factory object vary and depend on the design of the individual factory object. The events that you can map to functions appear automatically within the InControl Events Editor.

Note An event does not appear in the Events Editor if InControl does not support the data types of all the parameters that the event uses.

To map a new function to an event:

1. Click **Events** on the **Edit** menu. The Events Editor opens.
2. Locate the event and enter the function name in the adjacent Function column.
3. Click **OK** and the system prompts you to add a new function.
4. When you confirm, the **New** dialog box appears and you can select RLL or STL for the function code type.
5. Click **OK** and the editor adds the new function to the Function folder in the Project window.
6. Open the new function and enter the code.

To map an existing function to an event:

1. Click **Events** on the **Edit** menu. The Events Editor opens.
2. Locate the event and in the Function column, type the name of the function adjacent to the event.

   You can also click the Browse button to display the existing functions.

   ...

   All functions that are valid for use with the event are displayed.
3. If you want to examine or modify the function code, double-click the function name and the editor opens.

Defining the Function

The function that you map to the event is identical in form to any other InControl function, with the following exceptions:

- The first parameter must be an InOut parameter.
- The data type for the first parameter must be the same as the data type of the factory object used with the function. The data types for all other parameters must match those defined by the event.

When InControl automatically generates the parameters, it automatically assigns the first parameter the name ThisControl. You can change the name if you prefer.

You can map the same function to more than one event. In addition, InControl allows you to map a function to events in other factory objects.

WARNING! If an event triggers an inappropriate function call there is the potential risk of unpredictable operation by the controller, which can result in death or injury to personnel and/or damage to equipment.

Referencing InControl Factory Objects

For all ActiveX controls (including controls not designed as InControl factory objects), the properties and methods that can be mapped to an existing IEC-61131 data type are added as symbols to the Symbol Manager. You can monitor or modify these symbols from the InControl Watch Window, as well as from your application program.

To reference these symbols, use the following naming format for properties:

`<FOE name>.<property name>`

For example, to reference variables called Temp and Error in the FOE named TempControl, use

`TempControl.Temp` and

`TempControl.Error.`

For properties with parameters, use the following format:

`<FOE name>.<property name> (parameter list).`

Use an Assignment statement to access a property with parameters just as you do with a function. For example:

```
FOE1.param(3):=100;
```

Use the following calling format for methods:

`<FOE name>.<method name> (parameter list)`

For example, to call a method named Square in the FOE named Math, which takes one parameter, use:

```
Math.Square(6);
```

Note If a program (RLL, SFC, FOE, or Structured Text) attempts to write a value to a read-only variable of any ActiveX control, the attempt is ignored.

FOE variables are frozen at their last value when the FOE is set to the Stop mode. For example, the InCascade variable used in the PID FOE may be TRUE even when its associated loop is not running. When you are monitoring a variable, be sure to monitor the program Mode system variable as well, in order to verify that the contents of the variable are correct. Except for forced values, FOE values are initialized when they are downloaded, even during a smart download. Forced values remain forced during a smart download. Retentive values of the FOE are reloaded with their last saved values when the runtime engine is configured to restart automatically (Last, Pause, Run mode) after a system reboot.

Note For help in designing/developing FOEs, contact your distributor to obtain the InControl Factory Object Toolkit.

C H A P T E R 8

Running a Project

This chapter describes the InControl runtime environment: toolbar items, menu options, screen fields, etc.

Contents

Selecting Runtime Options

After designing the application programs within the InControl development environment, you execute the programs within the runtime environment. Programs go through three phases as they enter the runtime environment:

- Validation—a program is checked for syntax errors and compiled.
- Download—a program is loaded to the runtime engine.
- Run—a program is executed by the runtime engine.

You can validate, download, and run an individual program or an entire project. One InControl hardware unit supports one instance of the runtime engine, which can run one project at a time. However, you can download projects to runtime engines that are running on multiple hardware units.

For more information about running multiple projects, see "Running Multiple Projects" in the InControl System Administration chapter.

You can select all the InControl runtime options from the menu bar or from the Runtime toolbar, which appears when you have connected to the runtime engine. This chapter describes how to use these tools based on selections that you make from the menu bar. To avoid confusion, only one method is described in this manual.

Runtime Toolbar Commands

Runtime Menu Command	**Toolbar Icon**	**Description**
Connect / Disconnect	▦	Connects the Development environment to the runtime engine. The engine runs continually as a Windows service, and whether it actually executes a program as it runs, depends on its mode of operation (Run, Stop, etc.). When the runtime engine is connected, the icon is depressed and the option is "Disconnect," which disconnects the Development environment from the runtime engine. If you close (exit) the Development environment, the runtime engine continues to run.
Configure	n/a	Displays the **Offline Runtime Engine Properties** dialog box if not connected to the runtime engine. Displays the **Online Runtime Engine Properties** dialog box if connected to the runtime engine.
Report Status	n/a	Provides runtime engine status data, such as current project, time stamp, scan time, mode, processor utilization, faulted programs, I/O faults, etc. This data appears in the Output window and the Wonderware Logger.

Runtime Menu Command	Toolbar Icon	Description
Clear Faults	n/a	Sets faulted programs to Pause mode, clears I/O faults, and clears runtime engine error status bits, such as RTEngine.ScanOverrun.
Validate Project	✓	Validates all programs in a project. All modified programs are saved to the hard disk
Download Project [1]	⏬	Download all programs in a project to the Runtime Engine. Modified programs are saved to the hard disk. Programs are validated if necessary.
Upload Project Values	n/a	For all programs, replace defined initial values (for all local and global variables) with current values in the runtime engine. Does not upload I/O variables, arrays, or values that you cannot define during configuration, e.g., the Mode symbol. The Output window displays data that is uploaded.
Run Project [1]	▶	Run all programs in a project. Programs are validated and downloaded if necessary. All modified programs are saved to the hard disk.
Pause	⏸	Pauses all programs that are currently being run by the runtime engine. The I/O continues to be updated.
Single Scan	1▷	Executes a single scan of the runtime engine. I/O is updated, then all programs in a project that are currently downloaded to the runtime engine are executed one scan. Can only be done while runtime engine is paused.
Stop	■	Stops all programs that are currently being run by the Runtime Engine. Programs are unloaded from memory. The I/O goes to the state defined in the configuration for each I/O board.
Validate Program	✓	Validates selected program. If program was modified, it is saved to the hard disk.
Download Program [2]	⏬	Downloads selected program to the Runtime Engine. If necessary, the program is validated. If program was modified, it is saved to the hard disk.
Upload Program Values	n/a	For currently selected program, replace defined initial values of local variables with current values in the runtime engine. Does not upload arrays, or values that you cannot define during configuration, e.g., the Mode symbol. The Output window displays data that is uploaded.
Run Program [2]	▶	Runs the selected program. Program is validated and downloaded if necessary. If program was modified, it is saved to the hard disk.

Runtime Menu Command	Toolbar Icon	Description
Pause Program	⏸	Pauses a program that is currently being run by the Runtime Engine. The I/O continues to be updated.
Single Scan Program	▷	Executes a single scan of the program. I/O is updated and then the selected program is executed one scan. Can only be done while the program is paused.
Stop Program	■	Stops a program and unloads it from memory. I/O is unaffected.

1 Programs and I/O drivers that have been excluded from the project load on their property sheets are not downloaded.
2 Any configured I/O drivers are also downloaded to the runtime engine.

Connecting to the Runtime Engine

The development environment must be connected to the runtime engine before you download or run a program/project.

Note The runtime engine is a Windows service and starts automatically when you power up the hardware unit.

The runtime engine to which you connect can be located on either the local or a remote computer node. You specify the node in the Runtime Engine Properties dialog box, described in "Configuring the Runtime Engine" in the "InControl System Administration" chapter. To check the current target platform, right-click RTEngine in the Project window and select Properties.

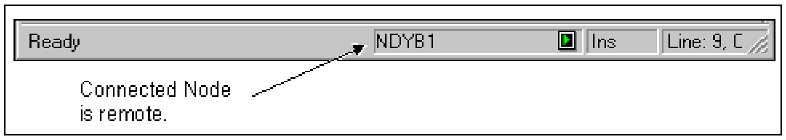
To connect to the runtime engine:

- On the **Runtime** menu, click **Connect**. After you connect, this option toggles to **Disconnect**.

You can also click **Connect Runtime Engine** on the runtime toolbar to connect to the runtime engine.

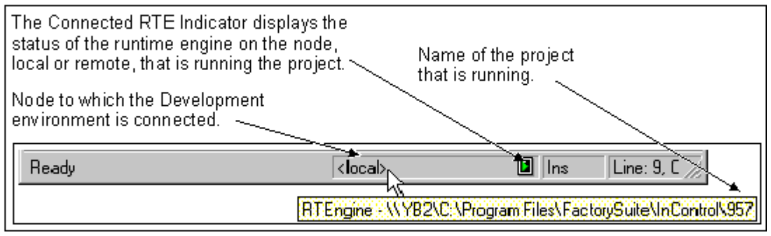
Checking the Connected Node

After connecting to the runtime engine, you can check the Status Bar and verify the computer node to which the Development environment is connected. In the following figure, the runtime engine is running on a node called NDYB1.



Identifying the Connected Node

If you place the cursor over the node name, the Tip help displays the name of the project that is running. In the following figure, the project called 957, which was developed on a remote node called YB2, is running on the local node.

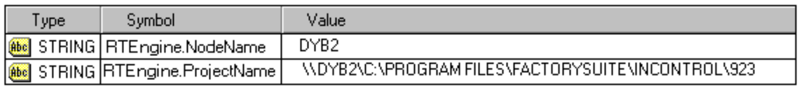

The Connected RTE Indicator displays the status of the runtime engine on the node, local or remote, that is running the project.
Name of the project that is running.
Node to which the Development environment is connected.
Ready
<local>
Ins
Line: 9, C

RTEngine - \\YB2\C:\Program Files\FactorySuite\InControl\957

Identifying the Running Project

You can also check the node and project by reading the value of these two runtime engine symbols: NodeName and ProjectName. Use these symbols to pass node and project names to another InControl program or to an HMI.

For example, to display the name of a project (called 923) and the node where it is executing (called DYB2), in the Watch Window, add the following symbols to the Watch Window: RTEngine.ProjectName and RTEngine.NodeName. The following figure shows the symbols as they appear in the Watch Window.

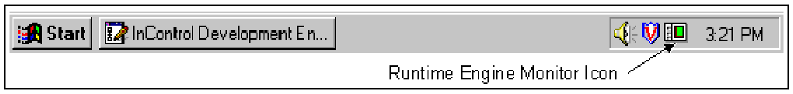


Type	Symbol	Value
STRING	RTEngine.NodeName	DYB2
STRING	RTEngine.ProjectName	\\DYB2\C:\PROGRAM FILES\FACTORYSUITE\INCONTROL\923

Displaying Project Information

Using the Runtime Engine Monitor

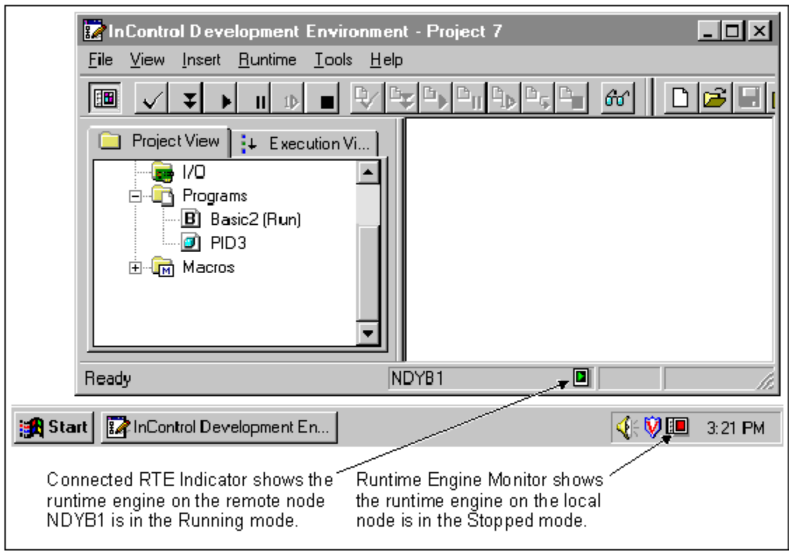
The runtime engine monitor is associated with the runtime engine on the local computer. You can use the monitor to send commands to and check the status of the local runtime engine. The runtime engine monitor icon is located in the task bar, as shown below.



Location of Runtime Engine Monitor Icon

Note InControl supports only one instance of the runtime engine on a computer.

Do not confuse the runtime engine monitor for the local node with the connected RTE indicator for the runtime engine to which the Development environment is connected. The runtime engine monitor, which is located on the task bar, is associated only with the local runtime engine. The connected RTE indicator, which is located on the InControl Status Bar, is associated with the runtime engine to which the Development environment is connected. This could be either the local node or a remote node.



Connected RTE Indicator shows the runtime engine on the remote node NDYB1 is in the Running mode.

Runtime Engine Monitor shows the runtime engine on the local node is in the Stopped mode.

Location of Connected RTE Indicator

Using the Runtime Engine Icons

The runtime engine monitor icon and the connected RTE indicator for the runtime engine to which the Development environment is connected have the following color codes:

- Red (square) indicates the Stop mode.
- Yellow (pause) indicates the Pause mode.
- Green (arrow) indicates the Run mode.

When variables are forced, the runtime engine monitor icon has a lock symbol:

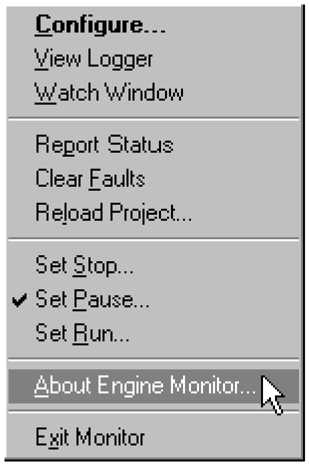
InControl indicates that a warning or error message has been sent to the Output window and the Wonderware Logger with a yellow diamond. This icon also appears when a program enters the Fault mode or when the RLL MSGW or Structured Text MSGWND functions execute.

InControl indicates a fault condition with the following symbol.

For information about clearing these fault conditions, see "Clearing Runtime Engine Fault Mode " in the "InControl System Administration" chapter.

Using the Runtime Engine Monitor Commands

You can place the cursor over the runtime engine monitor icon and right-click. The menu shown below appears. The check mark indicates the current mode.



The commands are described in the table that follows.

Runtime Engine Monitor Menu

Menu Options	Description
Configure	Displays the **Online Runtime Engine Properties** dialog box, described in "Setting Scan Times" in the "InControl System Administration" chapter).
View Logger	Displays the Wonderware Logger, which keeps a record of runtime messages. These messages also appear in the Output window. For more information, see "Checking the Wonderware Logger."
Watch Window	Displays the Watch window. This is a stand-alone version of the Watch window and it is not necessary to open the Development environment to access it. For more information about the standalone Watch window, see "Using the Stand-Alone Watch Window."
Report Status	Provides runtime engine status data, such as current project, time stamp, scan time, mode, processor utilization, faulted programs, I/O faults, etc. This data appears in the Output window and the Wonderware Logger.
Clear Faults	Sets faulted programs to Pause mode, clears I/O faults, and clears runtime engine error status bits, such as RTEngine.ScanOverrun.
Reload Project	Loads the last project that was successfully downloaded, including any changes. A stopped program is not reloaded. I/O goes to the state defined in the configuration for each I/O board.
Set Stop	Stops all programs that are running. The I/O goes to the state defined in the configuration for each I/O board. Programs are unloaded from memory. To run them again, click **Reload Project** on the **Runtime Engine Monitor** menu and then click **Set Run**.
Set Pause	Pauses all programs that are running. To run them you can click **Run** on the **Runtime Engine Monitor** menu or **Run Project / Program** on the **Runtime** menu of the standard menu bar in the Development environment.
Set Run	Run all programs currently loaded in the runtime engine.
About Engine Monitor	Displays the **About Runtime Engine** dialog box.
Exit Monitor	Removes the icon from the screen and puts the runtime engine into Stop mode.

Running/Exiting the Runtime Engine Monitor

The runtime engine monitor appears in the Startup directory and begins running automatically if the runtime engine service is running when you log on to the Windows operating system. If you exit the runtime engine monitor, follow one of the procedures below to restart it.

To start the runtime engine monitor from the menu bar:

Connect to the runtime engine and click **Download Project (Download Program)** or **Run Project (Run Program)** on the **Runtime** menu.

To start the runtime engine monitor from the Taskbar:

Click **Start** on the **Taskbar**, then click **Runtime Engine Monitor** under **InControl**.

Starting the runtime engine service also starts the runtime engine monitor.

When you exit the runtime engine monitor, any programs that are running are stopped. The icon is always displayed when programs are loaded or running and you are logged on to the Windows operating system.

To exit the runtime engine monitor:

1. Right-click the runtime engine monitor icon to display the menu.
2. Click **Exit Monitor**. If programs are running, you are prompted to confirm that they be stopped.

   The icon is removed.

WARNING! In the unlikely event that the runtime engine terminates abnormally, the runtime engine monitor icon may still remain visible, although it will display the Fault mode icon. With the runtime engine stopped, factory output devices may operate unpredictably with the potential risk of death or injury to personnel and/or damage to equipment. It is highly recommended that you hardwire alarms to indicate abnormal operation by the factory process.

Validation and Download

You can validate a project/program, regardless of the operating system used by the target runtime engine. Note that you can change the runtime engine target before validating (right-click the runtime engine icon in the Project Window to display the Properties dialog box). This is useful when you develop and test a project on a computer using the Windows operating system but intend to download and run the project on a computer that uses a different operating system.

System operation is slightly different if you validate and download an individual program instead of a project. When you work at the project level, all programs within the project are affected. When you work at the program level, actions you take on an individual program act on that program and may possibly affect other programs within the project. Therefore, procedures for validation and downloading for programs and projects are described separately.

- Project validation is described in "Validating a Project."
- Program validation is described in "Validating an Individual Program."

Validating a Project

You can validate all the programs within a project any time during the project development. It is not necessary for the Development environment to be connected to the runtime engine when you validate a project. This allows you to validate a project that you are developing on a Windows system but intend to run on a computer using another operating system.

Typically, you use the Project Validation option to check for syntax errors in programs and to compile them without downloading and running them.

To validate a project:

1. Click **Validate Project** on the **Runtime** menu. The **Validate Project** dialog box appears.
2. Select the appropriate validation option, as described below. The programs are saved as needed and validated.

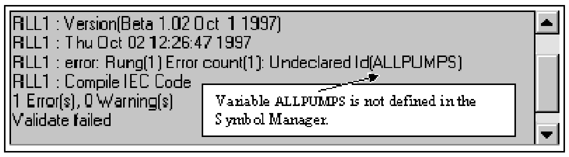
- To validate all programs, whether or not they have been changed, select **Full Validate**.
- To validate only programs that have been changed, select **Smart Validate**.

Optional. Check the **Enable Debug** checkbox to enable any Structured Text BREAK functions that have been programmed, or to use the Breakpoint feature in Structured Text programs. The BREAK function causes an SFC or STL program to pause when program flow encounters the BREAK. You must also check the **Enable Debug** checkbox if you want to single-step a program. Note that execution time is slower when the Debug feature is enabled.

Optional. Check the **Create Executable Archive** checkbox to create a file containing the compiled project. You can use the file as an archive or copy it to a node where you can run it.

For more information about using the archive file, see "Transferring/Archiving Project Data" in the InControl System Administration chapter.

3. Check the Output window for messages. A system message reporting a failed validation and showing the location of the errors is shown in the following figure:



```
RLL1 : Version(Beta 1.02 Oct 1 1997)
RLL1 : Thu Oct 02 12:26:47 1997
RLL1 : error: Rung(1) Error count(1): Undeclared Id(ALLPUMPS)
RLL1 : Compile IEC Code
1 Error(s), 0 Warning(s)
Validate failed
```

Variable ALLPUMPS is not defined in the Symbol Manager.

Output Window after Project Validation

If a program fails validation, refer to the error messages in the Output window and make the appropriate changes in your program code. Note that all the messages may not be visible, and you may need to scroll back to view additional messages.

Note The Smart Validate feature does not remove any symbols from the runtime engine when they have already been downloaded. Symbols are removed only when you set a Project to the Stop mode or you do a Full Restart. Therefore symbols that you deleted in the Symbol Manager are still visible in the Watch window. You can modify them in the Watch window although this will have no effect on the execution of your logic.

Downloading a Project

When you download a project, you must connect to the runtime engine first.

To download a project:

1. Connect to the runtime engine.
2. Click **Download Project** on the **Runtime** menu.

   The **Download Project** dialog box appears.
3. Select the appropriate download option, as described below, and click **OK**. The programs are saved and validated, if needed, and downloaded.

- The following actions require a Full Reload:

  Downloading a project configuration (I/O configuration or runtime engine configuration) that has not been downloaded before.

  Editing an I/O Configuration.

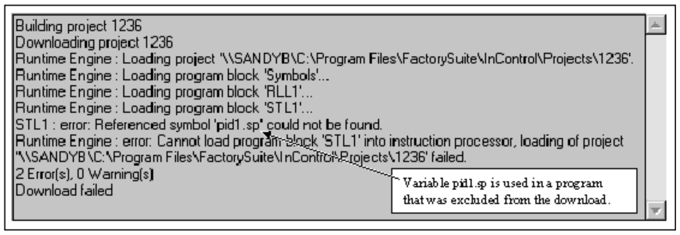
  Making these modifications to variables, including user-defined types, that have already been downloaded: changing the data type, the lower element of an array, or the bit index information.

  Changing the offline runtime engine configuration.
- If project configuration data has already been loaded during a previous download and you do not want to interrupt programs that are currently running, select **Smart Load**. All new programs are validated and downloaded. Programs that are running, and that have been modified, are paused and replaced by the modified versions; these programs remain paused. Other programs, which are running, are unaffected. Stopped programs are validated and downloaded.

  If a program is running and you download a modified version of the program, the original program is paused, even if the new version fails to download. If the download is successful, the asterisk by the program name in the title bar is removed.

  If you do not want a program to be paused after a new version is downloaded, click **Run Project**, not **Download Project**, on the **Runtime** menu. The program is downloaded and set to the Run mode if the download is successful.

4. Check the Output window for messages. A system message reporting a failed download and showing the location of the errors is shown in the following figure:



```
Building project 1236
Downloading project 1236
Runtime Engine : Loading project '\\SANDYB\C:\Program Files\FactorySuite\InControl\Projects\1236'.
Runtime Engine : Loading program block 'Symbols'...
Runtime Engine : Loading program block 'RLL1'...
Runtime Engine : Loading program block 'STL1'...
STL1 : error: Referenced symbol 'pid1.sp' could not be found.
Runtime Engine : error: Cannot load program block 'STL1' into instruction processor, loading of project
'\\SANDYB\C:\Program Files\FactorySuite\InControl\Projects\1236' failed.
2 Error(s), 0 Warning(s)
Download failed
```

Variable pid1.sp is used in a program that was excluded from the download.

Output Window after Project Download

If a download fails, refer to the error messages in the Output window and make the appropriate changes in your program code. Note that all the messages may not be visible, and you may need to scroll back to view additional messages.

Note If you have included any code for testing or simulating your process and it writes to I/O input variables, a warning message appears. You can change this to an error message or prevent its appearing altogether. See "Displaying Compiler Warnings" in the "InControl System Administration" chapter.

If you download a project using the **Smart Load** option, local and global variables that were already downloaded retain their current values and are not reinitialized. SFCs are restarted.

Note The Smart Load feature does not remove any symbols from the runtime engine when they have already been downloaded. Symbols are removed only when you set a Project to the Stop mode or you do a Full Restart. Therefore symbols that you deleted in the Symbol Manager are still visible in the Watch window. You can modify them in the Watch window although this will have no effect on the execution of your logic.

You can exclude a program from project downloads. Right-click a program in the Project View and select **Exclude**. This is useful for simulation programs that you do not normally need to execute, but which you want to keep with the project.

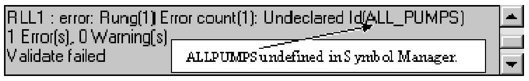
To set the programs in the project to the Run mode, see "Running a Project."

Validating an Individual Program

You can validate a program, regardless of the operating system used by the target runtime engine. In addition, you can change the runtime engine target (right-click the runtime engine icon in the Project Window to display the **Properties** dialog box). This is useful when you develop and test a program on a computer using the Windows operating system but intend to download and run the program on a computer that uses a different operating system.

To validate a program:

1. Click **Validate Program** on the **Runtime** menu. The **Validate Program** dialog box appears.
2. Optional. Check the **Enable Debug** checkbox to enable any Structured Text BREAK functions that have been programmed, or to use the Breakpoint feature in Structured Text programs. The BREAK function causes an SFC or STL program to pause when program flow encounters the BREAK. You must also check the **Enable Debug** checkbox if you want to single-step a program. Note that execution time is slower when the Debug feature is enabled.
3. Click **OK**. The program is saved, if necessary, and validated.
4. Check the Output window for messages. A system message reporting a failed validation and showing the location of the errors is shown in the following figure:



```
RLL1 : error: Rung(1) Error count(1): Undeclared Id(ALL_PUMPS)
1 Error(s), 0 Warning(s)
Validate failed
```

ALLPUMPS undefined in Symbol Manager.

Output Window after Program Validation

If a program fails validation, check the error messages in the Output window and make the appropriate changes in your program code. Note that all the messages may not be visible, and you may need to scroll back to view additional messages.

Downloading an Individual Program

When you download a program, you must connect to the runtime engine first. Any configured I/O drivers are also downloaded along with the program.

To download a program:

1. Connect to the runtime engine.
2. Download any other POUs (functions, function blocks, other programs, etc.) on which the program depends. Otherwise, the program will fail to download.
3. Click **Download Program** on the **Runtime** menu. The **Download Program** dialog box appears.
4. Select the appropriate download option and click **OK**. The program is saved and validated, if needed, and downloaded.

- The following actions require that you select **Reload Runtime Engine**. Note that this option unloads all currently loaded programs and I/O drivers.

  Downloading a project configuration (I/O configuration or runtime engine

  configuration) that has not been downloaded before.

  Editing an I/O Configuration.

  Making these modifications to variables, including user-defined types, that

  have already been downloaded: changing the data type, the lower element of an array, or the bit index information.

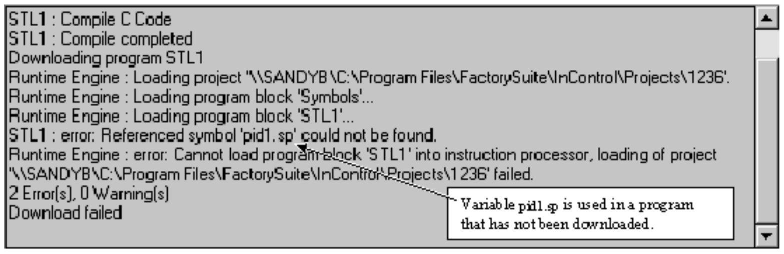
  Changing the offline runtime engine configuration.

- If project configuration data (I/O configuration or runtime engine configuration) has already been loaded during a previous download and you do not want to interrupt programs that are currently running, select **Reload**.

  If a program is running and you download a modified version of the program, the original program is paused, even if the new version fails to download. If the download is successful, the asterisk by the program name in the title bar is removed.

  If you do not want program execution to be interrupted, click **Run Program**, not **Download Program**, on the **Runtime** menu. The program is downloaded and it begins to execute upon completing the download.

5. Check the Output window for messages. A system message reporting a failed download and showing the location of the errors is shown in the following figure:



```
STL1 : Compile C Code
STL1 : Compile completed
Downloading program STL1
Runtime Engine : Loading project "\\SANDYB\C:\Program Files\FactorySuite\InControl\Projects\1236'.
Runtime Engine : Loading program block 'Symbols'...
Runtime Engine : Loading program block 'STL1'...
STL1 : error: Referenced symbol 'pid1.sp' could not be found.
Runtime Engine : error: Cannot load program block 'STL1' into instruction processor, loading of project
"\\SANDYB\C:\Program Files\FactorySuite\InControl\Projects\1236' failed.
2 Error(s), 0 Warning(s)
Download failed
```

Variable pid1.sp is used in a program that has not been downloaded.

Output Window after Program Download

If a program download fails, check the error messages in the Output window and make the appropriate changes in your program code.

Note If you have included any code for testing or simulating your process and it writes to I/O input variables, a warning message appears. You can change this to an error message or prevent its appearing. See "Displaying Compiler Warnings" in the "InControl System Administration" chapter.

If you reload a program that was previously downloaded, local and global variables that were already downloaded retain their current values and are not reinitialized. SFCs are restarted.

To set the mode of a program to Run, see "Running an Individual Program."

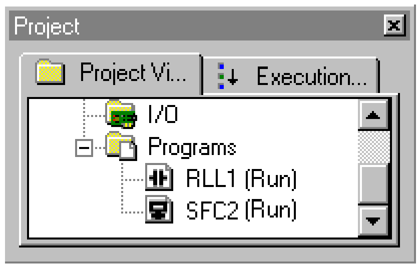
Project/Program Execution

System operation is different if you run a project instead of an individual program. Therefore, procedures for running programs and projects are described separately.

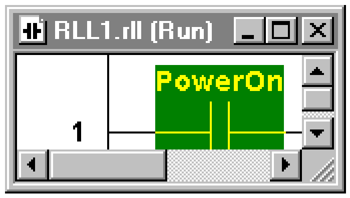
Running a Project

After the programs in the project are successfully validated and downloaded to the runtime engine, their mode is set to Pause. Note that if you click **Run Project** instead of **Download Project**, the programs are set to the Run mode after they are downloaded.

InControl indicates a program's mode in the Project window as shown in the following figure (all programs are running).



When a program is open in an editor, InControl also indicates the program's mode in the program's Title bar, as shown in the following figure.



WARNING! Running a program that has not been thoroughly tested on a system connected to field devices may cause unpredictable operation by the devices.
Unpredictable operation by field devices may cause injury or death and/or damage to equipment. It is highly recommended that you test your program before running it on a system that controls a factory process. Verify the correct operation of every program element or line of code. Note that some error conditions are not detected until run time and these may disable some or all of your program logic.
For more information about troubleshooting a program, see "Debugging a Program."

To run the programs in a project:

1. Click **Run Project** on the **Runtime** menu.

   The **Run Project** dialog box appears.

2. Select the appropriate run option.

- If the project configuration (I/O configuration or runtime engine configuration) has changed, select **Full Restart**. Note that all programs are restarted, including SFCs that have completed execution. The I/O goes to the state defined in the configuration for each I/O board.
- To reload only programs in a project that you have modified, select **Smart Start**. This option has no effect on other programs that are running, but you can choose it only if you have made no changes to the I/O configuration or the runtime engine configuration. Variables in programs are not initialized except for those used by FOEs.
- To ignore changes made in any program, select **Continue**.

3. Click **OK** to confirm. All modified programs are saved, validated, downloaded, if needed, and set to RUN mode. An asterisk by the program name in the Project window or in the window title bar indicates that the program has not been saved, and/or is different from the copy running in the runtime engine.

When the programs in a project change to the Run mode, the connected RTE indicator, located in the InControl status bar, turns green.

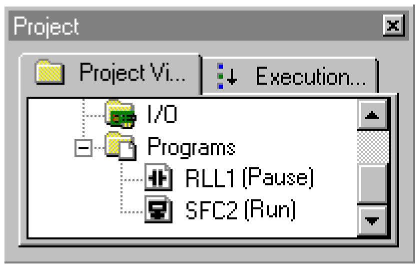
Note Only one project can run on the runtime engine at a time. However, you can create/edit programs in one project while the programs in another project are running. In addition, you can run projects on runtime engines installed on other computers, as described in "Running Multiple Projects" in the "InControl System Administration" chapter.

For a description of the order in which programs are executed, see "Project/Program Execution Order."

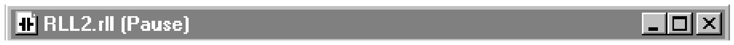
Running an Individual Program

After a program is successfully validated and downloaded to the runtime engine, the mode of the program is set to Pause. Note that if you click **Run Program** instead of **Download Program**, the program is set to the Run mode after it is downloaded.

InControl indicates a program's mode in the Project window as shown in the following figure (RLL1 is paused; SFC2 is running).



When a program is open in an editor, InControl also indicates the program's mode in the program's Title bar, as shown in the following figure.



WARNING! Running a program that has not been thoroughly tested on a system connected to field devices may cause unpredictable operation by the devices.
Unpredictable operation by field devices may cause injury or death and/or damage to equipment. It is highly recommended that you test your program before running it on a system that controls a factory process. Verify the correct operation of every program element or line of code. Note that some error conditions are not detected until run time and these may disable some or all of your program logic.
For more information about troubleshooting a program, see "Debugging a Program."

To run a program:

1. Select the program (click the program name in the Project window, or make it the active program in the editor) and click **Run Program** on the **Runtime** menu.

   The **Run Program** dialog box appears.
2. Select the appropriate run option.

- If the project configuration (I/O configuration or runtime engine configuration) has changed, select **Restart Runtime Engine**. This option stops all programs that are running and only restarts the program being downloaded, plus I/O. The I/O goes to the state defined in the configuration for each I/O board.
- If you want the program variables to keep their current values, select **Restart**. Variables in programs are not initialized except for those used by FOEs. This option has no effect on any other programs that are running.
- To continue executing the version of the program already on the runtime engine, select **Continue**. Any modifications that were made in the program are saved in the program file, but are not downloaded.

3. Click **OK** to confirm.

When the program changes to the Run mode the Runtime Engine Monitor icon, located in the task bar, is green to indicate that it is in the Run mode. Any configured I/O drivers are also downloaded to the runtime engine and begin to run.

Note You can edit a program, or other programs, while the program is running. In addition, you can run projects on runtime engines installed on other computers, as described in "Running Multiple Projects" in the "InControl System Administration" chapter.

For a description of the order in which programs are executed, see "Project/Program Execution Order."

Stopping a Project

When you stop a project, all programs are unloaded from the runtime engine and the I/O output points are disabled. The individual I/O drivers determine the actual state of the outputs.

To stop project execution:

- Click **Stop** on the **Runtime** menu,

  OR

- Click **Stop Runtime Engine** on the Runtime toolbar:

  OR

- Click **Set Stop** on the menu for the runtime engine monitor icon.

  The **Stop Runtime Engine** dialog box appears.

When you click **OK** all programs are stopped. Because they are unloaded from the runtime engine, you must download them from the Development environment again before you can run them. Either click **Run Project** on the **Runtime** menu or click **Run Project** on the Runtime toolbar. You can also click **Reload Project** on the runtime engine monitor menu.

Note The **Set Run** option on the menu for the runtime engine monitor icon runs only projects that have been paused, not projects that have been stopped.

Program variables are frozen at their last value when a program is set to the Stop mode. For example, the InCascade variable used in the PID FOE may be TRUE even when its associated loop is not running. When you are monitoring a program variable, be sure to monitor the program Mode system variable as well, in order to verify that the contents of the variable are valid.

InControl automatically sets the value of the runtime engine system variable RTEngine.Mode to STOP (1) when the runtime engine is stopped. You can change the mode of a project from the Watch window or other external program by writing one of the following values to the RTEngine.Mode system variable: PAUSE (2), SCAN (3), RUN (5), or FAULT (6). However, you cannot write STOP (1) or any other values to RTEngine.mode.

Stopping a Program

When you stop a program, the program is unloaded from the runtime engine. The I/O points are not affected.

To stop program execution:

1. If the program is open in the editor window, select it.
2. Click **Stop Program** on the **Runtime** menu,

   OR

   Click **Stop Program** on the Runtime toolbar:

   The **Stop Program** dialog box appears.

When you click **OK** the program is stopped. Because it is unloaded from the runtime engine, you must download it again before you can run it. Either click **Run Program** on the **Runtime** menu or click **Run Program** on the Runtime toolbar:

Note The **Set Run** option on the menu for the runtime engine monitor icon runs only projects that have been paused, not projects that have been stopped.

You can also stop a program by selecting it in the Project View before clicking **Stop Program**. If another program is open and selected in the editor window, it is the program that is selected in the Project View that is stopped.

Program variables are frozen at their last value when a program is set to the Stop mode. For example, the InCascade variable used in the PID FOE may be TRUE even when its associated loop is not running. When you are monitoring a program variable, be sure to monitor the program Mode system variable as well, in order to verify that the contents of the variable are valid.

The value of the program Mode system variable is set to 1 when the program is stopped.

Project/Program Execution Order

You can specify the order within the scan in which programs are executed, and you assign a priority level to each program: Normal Scan and Low Priority.

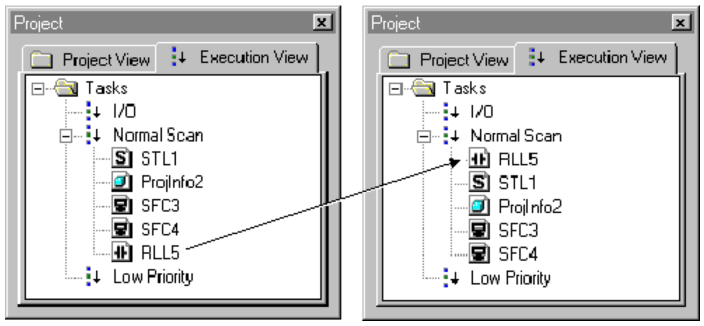
For more information about the timeline and program execution, see "Runtime Engine Timeline" in the "InControl System Administration" chapter.

Setting Program Order in the Execution View

You can specify the order within the scan in which programs are executed if you download the programs as a project. When you select the Execution View in the Project window, programs are displayed in the order that they are executed.

To change the execution order:

1. Display the Execution View.
2. Click a program name.
3. Drag the program to another location in the list of programs. In the following figure RLL5 has been moved above STL1.



Changing Execution Order

Note these guidelines.

- If you download programs individually, instead of as a project, the programs are executed in the order that you download them. If you later download the project, execution order is determined by the order of the programs in the Execution View.
- If you change the program order in the Execution View, you must download the project for that order to be maintained. Use either the **Smart Load** or the **Smart Start** option.
- If you download a project and then download a version of a program that is currently running in the project, execution order for the program is preserved. However, if you create a new program in the project and then download the new program, it goes to the bottom of the queue and runs after the other programs in the project.

Setting Program Priority in the Execution View

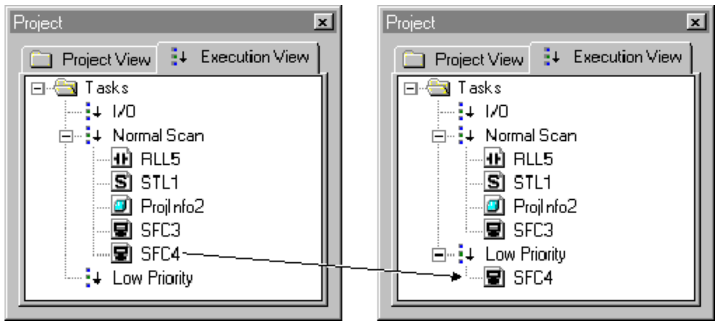
You can specify an execution priority for a program: Normal Scan or Low Priority. InControl runs a program set at Low Priority only if there is time remaining after all other tasks in the timeline have been done. During the low-priority time slice of a scan, as many programs are executed as possible, with the execution of one program resuming after the previously executed low-priority program has completed.

For more information about the timeline and how programs are executed, see "Runtime Engine Timeline" in the "InControl System Administration" chapter.

To change the execution priority:

1. Display the Execution View.
2. Click a program name.
3. Drag the program to the new position (Low Priority or Normal Scan) on the List of Tasks.

   In the following figure, SFC4 has been assigned Low Priority.



Changing Program Priority

Note No flags are set if a low-priority program fails to execute because the scan time has been set too low. You must include code in your program to check whether the program executes as needed.

Debugging a Program

Use the options described in this section to check the program as it is run.

Checking the Status Bar

The Status bar, illustrated in "Runtime Window" of the "InControl Environment" chapter, displays the status of the runtime engine that is running the project, the name of project downloaded to the runtime engine, the node to which the Development environment is connected, as well as other program information.

Checking the Program Mode

During the course of its execution, a program may do an illegal operation and enter the Fault mode.

A program fault is different from a runtime engine fault. For information about the program and runtime engine Fault modes, see "Clearing Program Fault Mode" in the "InControl System Administration" chapter.

If a program enters the Fault mode, "Fault" appears in the program's title bar and in the Project View. InControl sets the Mode system variable for the program to six, writes a message to the Output window and the Wonderware Logger, and superimposes the following symbol over the runtime engine monitor icon:

⚠

To restart a program in the Fault mode:

1. Clear faults. On the **Runtime Engine Monitor** menu, click **Clear Faults**. This sets the program to the Pause mode. If other programs are running, they continue to run and the runtime engine remains in the Run mode.

   You can also set the program to the Run or Pause mode to clear faults.
2. Set the program to the Run mode.
3. If the program returns to the Fault mode, you need to check program logic and possibly the integrity of your hardware. If you change the program, download it again and run it.

Note InControl writes a message to the Output window and the Wonderware Logger when a program enters the Fault mode.

You can use one of the runtime engine functions to report runtime engine status and clear faults for a program or a project. For a program, use the following syntax:

```
RTEngine.ExecProgramCommand ("program name", <command>)
```

where **<command>** equals ReportStatus to report runtime engine status information, and **<command>** equals ClearFaults to clear program faults.

For a project, use the following syntax:

```
RTEngine.ExecProjectCommand (<command>)
```

where **<command>** equals ReportStatus to report runtime engine status information, and **<command>** equals ClearFaults to clear program faults.

For more information about clearing faults, see "Clearing Fault Mode and Error Conditions" in the InControl System Administration chapter.

Example 1: to examine runtime engine status data and to report faults for a program called STL1, enter the following line in a program:

```
RTEngine.ExecProgramCommand("STL1",ReportStatus);
```

Be sure to use quotation marks around the program name.

Example 2: to clear faults for a project called Bldg_A, enter the following line in a program:

```
RTEngine.ExecProjectCommand(ClearFaults);
```

Note These methods do not clear a runtime engine Fault.

Single Scanning a Project/Program

The Single Scan option allows you to execute a single program or all programs in a project for one complete scan by the runtime engine. Execution follows the timeline that is described in "Runtime Engine Timeline" of the "InControl System Administration" chapter. Order of program execution is based on the order of the programs in the Execution View. A project/program must be in the Pause mode before you can execute a single scan.

To execute a single scan:

1. Set the project or program to the Pause mode.
2. Optional. Open a Watch window, and add the variables that you need to monitor as the programs run.
3. Click **Single Scan** (or **Single Scan Program**) on the **Runtime** menu. All paused programs enter the Run mode during the next scan by the runtime engine, remain in the Run mode during the scan, then reenter the Pause mode.

Using Breakpoints

You can pause program execution at a selected line of code in a Structured Text program. Program flow pauses before executing the selected line. For program flow to continue, you can run, unpause, single scan, or single step the program. You can place breakpoints at multiple lines of code, and when program flow continues, it pauses at the next breakpoint.

You must check the **Enable Debug** checkbox in either of the following two dialog boxes prior to validating the program before you can set a breakpoint in a Structured Text program. Checking one checkbox automatically checks the other.

- The **Validate Project** or the **Validate Program** dialog box. On the **Runtime** menu, click **Validate Program** to display this dialog box.
- The **Properties** dialog box for each program that you want to step. In the Project View, right-click the program and click **Properties** to display this dialog box.

A breakpoint operates in the same way as the Structured Text BREAK statement. However, you must remove the BREAK statement for the program or uncheck one of the **Enable Debug** checkboxes when you are finished troubleshooting the program. You can easily add and remove breakpoints at runtime by clicking **Toggle Breakpoint** and **Clear All Breakpoints** on the **Runtime** toolbar. Note that execution time is slower when the Debug feature is enabled.

Note Program flow does not stop at a breakpoint in a function if the function is running in the background.

Stepping a Program

The Step Program option allows you to execute a line of logic in a single program (RLL, SFC, or Structured Text). A program must be in the Pause mode before you can step it. You must check the **Enable Debug** checkbox in either of the following two dialog boxes prior to validating the program before you can step through these programs.

- The **Validate Project** or the **Validate Program** dialog box. On the **Runtime** menu, click **Validate Program** to display this dialog box.
- The **Properties** dialog box for each program that you want to step. In the Project View, right-click the program and click **Properties** to display this dialog box.

Each time you step the program, programs are executed as follows:

- For an RLL program, one rung of logic is executed.
- For a Structured Text program, one line of code is executed.
- For an SFC program, one line of code of every active Step is executed. All active Actions are executed completely.
- For an FOE, the Step Program option is not supported.

Note Single-stepping any program may cause execution order to differ from that of a normally running program. This is due to the continuous execution of I/O, other programs, and in the case of SFCs, Actions.

To step a program:

1. Select the program and set it to Pause mode.
2. Optional. Open a Watch window for the program and add any variables that you need to monitor as the program runs.
3. Click **Step Program** on the **Debug** menu. The program enters the Run mode during the next scan by the runtime engine, remains in Run mode for one scan, then reenters the Pause mode. Any I/O controlled by the program is updated when the program is stepped.

You can use a runtime engine function to single-step one or more programs. Use the following syntax:

```
RTEngine.ExecProgramCommand (“program name”,
   RTECommand.Step)
```

For example, to single-step three programs, called STL1, STL2, and RLL5, enter the following lines in another program that you are using to troubleshoot your code:

```
RTEngine.ExecProgramCommand(“STL1”,RTECommand.Step);
```

```
RTEngine.ExecProgramCommand(“STL2”,RTECommand.Step);
```

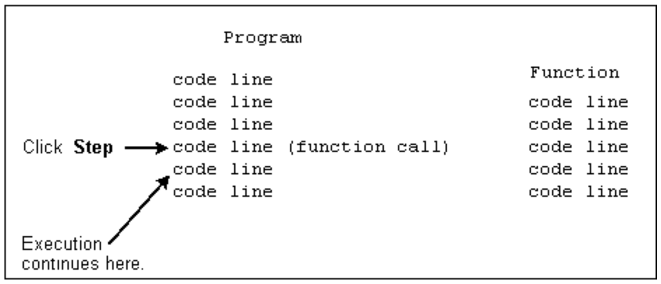
```
RTEngine.ExecProgramCommand(“RLL5”,RTECommand.Step);
```

Note A program cannot single-step itself. Always single-step one or more programs from another program. In addition, remember that a program must be in the Pause mode before you can single-step it. You can set a program to the Pause mode by writing to the program's Mode system variable.

WARNING! Specifying invalid values for the command parameter may cause unpredictable I/O operation with the potential risk of death or injury to personnel and/or damage to equipment. Use only the values listed above for the command parameter.

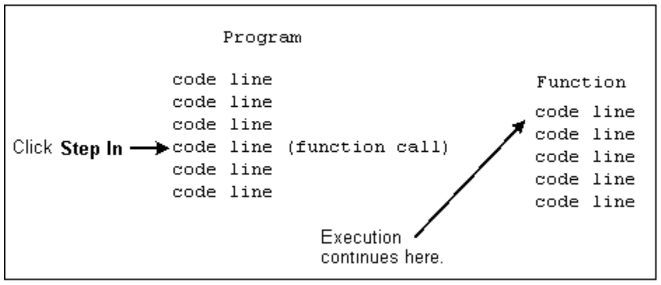
Use the **Step In** and **Step Out** commands on the **Debug** Menu to debug code in functions.

In the following example, program execution is paused at the line of code with the function call. If you click **Step** on the **Debug** menu, program flow continues at the line following the function call.



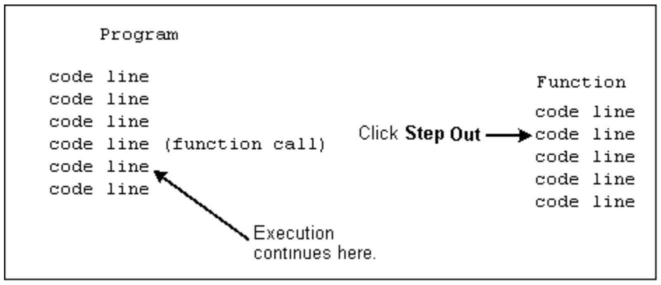
Step Example 1

In the following example, program execution is paused at the line of code with the function call. If you click **Step In** on the **Debug** menu, program flow continues at the first line in the function.



Step Example 2

In the following example, program execution is paused at a line of code within the function. If you click **Step Out** on the **Debug** menu, program flow continues at the line following the function call in the program making the function call.



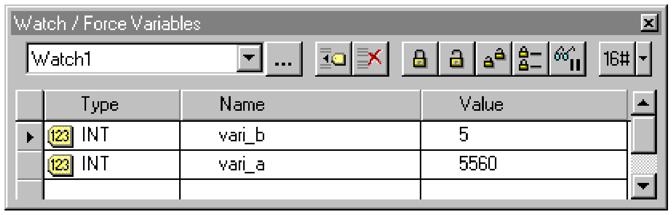
Step Example 3

Click **Show Call Stack** to display the current values of local symbols and parameters for a function that has been called. If multiple functions have been called, you can select the function that you need to monitor.

Monitoring Program Variables

The Watch window, shown in the following figure, displays variables and their status at runtime. You can open one Watch window for each project, and for each window you can create one or more tables of variables to be monitored.

Within a table, you can monitor variables from one or more programs, including local, global, and system variables, or combinations of each. You cannot monitor the values of variables or parameters used in a function or procedure.



Type	Name	Value
INT	vari_b	5
INT	vari_a	5560

Watch Window Displaying Two Variables

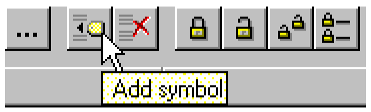
Display or hide the Watch window as needed by clicking **Watch/Force Variables** on the **View** menu or by clicking the **Watch window** tool on the Runtime toolbar:

You can also run a stand-alone version of the Watch window. It is not necessary to open the Development environment to access it. On the runtime engine monitor icon, click **Watch Window**. It is also available from the **Start** menu on the Windows task bar.

Adding a Variable to the Watch Window

To display a variable in the Watch window:

1. Click **Add Symbol** on the Watch window toolbar.

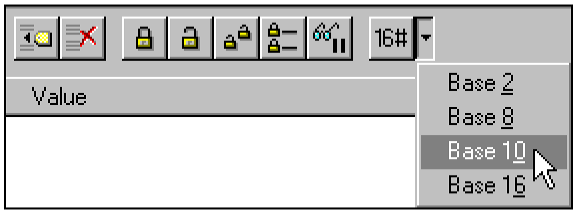


   The **Symbol Manager** dialog box appears.

2. Click the name of one or more variables to add. You can right-click a variable to view a read-only display of its properties.

3. Click **OK**. The selected variables appear in the Watch window.

For BYTEs, WORDs, DWORDs, SINTs, DINTs, REALs, and LREALs, you can choose to display the value in binary, octal, decimal, or hexadecimal formats. Click the name of the variable and then choose the format from the Format tool on the Watch window toolbar.



Watch Window – Changing Format

If you add a variable that has been defined as a constant to the Watch window, the variable appears as a dimmed value. You cannot modify the variable in the Watch window.

To view a variable that is local to a macro, use the following syntax:

```
<Parent SFC Program>.<Macro Step Name>.<Local Symbol Name>
```

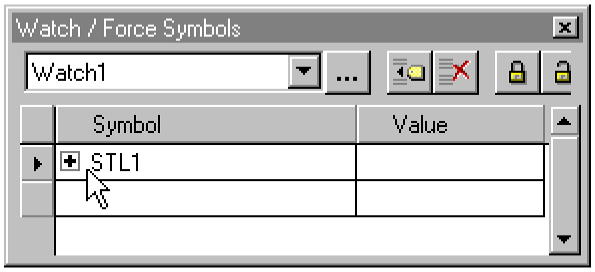
If a variable has an initial value in the Symbol Manager, the format in the Watch window matches the format in the Symbol Manager. For example, if the initial value is 2#1101_0110, then the default format in the Watch window is binary.

Adding Multiple Variables to the Watch Window

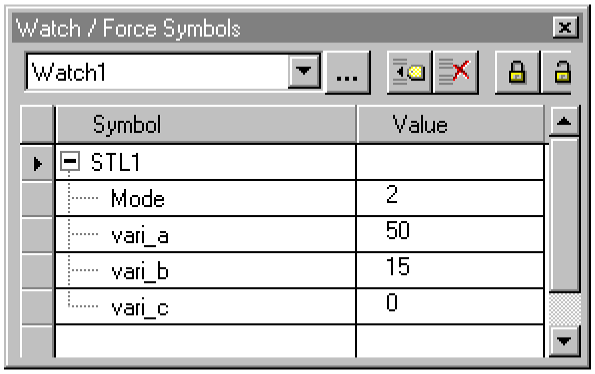
Instead of selecting variables individually to display in the Watch window, you can add variables that are related by group. For example, you can add all variables used in a program, or all the member variables in a structure.

To display all program variables in the Watch window:

1. Place the cursor in the **Symbol** field and type in the name of the program.
2. Press **Enter**. The program variables appear in a collapsed tree structure.



3. Expand the tree and all the variables of the program appear.



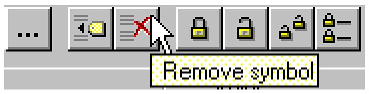
Symbol	Value
STL1	
Mode	2
vari_a	50
vari_b	15
vari_c	0

Watch Window - Adding Multiple Variables 2

Removing a Variable

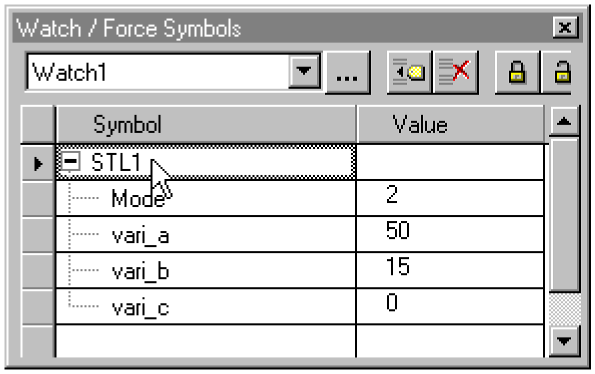
To remove a variable from the Watch window:

1. Click the name of the variable.
2. Click **Remove Symbol** on the Watch window toolbar:



The variable is removed from the Watch window and deleted from the .WCH file.

You cannot delete variables individually if you added them as a group. You must delete the entire goup by clicking the top-level of the group beforing clicking **Remove Symbol**.

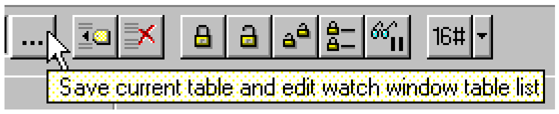


Symbol	Value
STL1	
Mode	2
vari_a	50
vari_b	15
vari_c	0

Adding a Table of Variables to the Watch Window

To add a table to the Watch window:

1. Click **Save Current Table** on the Watch window toolbar.



   The **Watch Tables** dialog box appears displaying the default table called **Watch1**.

2. Click **New** to add a table. When the **New** dialog box appears, enter a name for the new table and click **OK**. Click **OK** again to return to the Watch window.

3. Add the variables to the Watch window.

Any tables that you create are saved when you display another table or exit InControl. The tables are saved in the same directory where the project files are located.

To make a copy of an existing table, click **Save As** and enter a name for the copy in the **Save As** dialog box.

As an alternative to creating the tables within the Watch window, you can create an ASCII file with any text editor.

To create a table with an ASCII text editor:

1. Open the text editor.

2. Enter the names of the variables using the following formats.

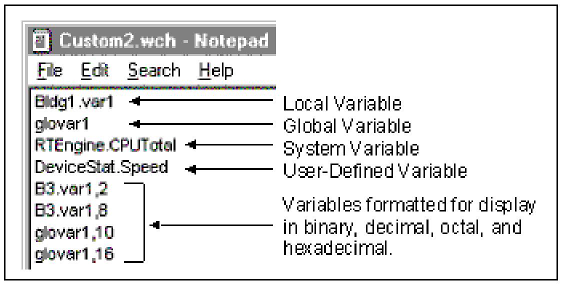
   For a local variable, use the program name, a period, and then the variable name. For the local variables in the example below, the program names are Bldg1 and B3.

   For a global variable, use only the variable name.

   For a runtime engine system variable, use the system name, RTEngine, followed by a period and the variable name.

   For a user-defined variable, use the name of the variable of the user-defined data type, followed by a period and the member name.

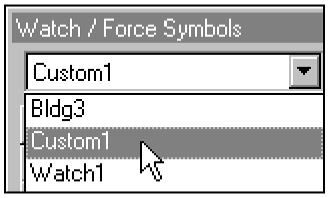
   The following figure shows an example.



```
Bldg1.var1
glovar1
RTEngine.CPUTotal
DeviceStat.Speed
B3.var1,2
B3.var1,8
glovar1,10
glovar1,16
```

Watch Window - Editing Table

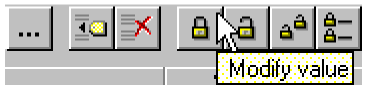
3. Save the file with the extension .WCH and place it in the project directory. The file name will appear in the table list with the other table names.



Modifying/Forcing a Variable

To modify or force a variable:

1. Click the name of the variable.
2. Click **Modify Value** on the Watch window toolbar.



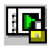
   The **Modify Value** dialog box appears.

3. To write a new value to the variable, enter the value (select TRUE or FALSE for a Boolean data type) and click **Write**. The new value is written to the variable. Note that the value may change as the program runs.

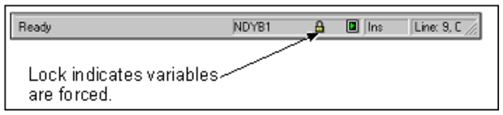
   To force a variable to a value, enter the value (select TRUE or FALSE for a Boolean data type) and click **Force**. The variable is forced to the new value, and the value does not change as the program runs until you force it to a new value or unforce it.

   When you enter a value, you can separate characters with underscores. For binary and hexadecimal numbers, the Watch window separates every four characters with an underscore to improve readability.

   When variables are forced, the runtime engine monitor icon has a lock symbol:



   The Status Bar also displays a lock symbol when variables are forced:



Note these additional points:

- If you force a bit that is indexed (described in "Assigning a Name to a Bit" in the "Defining Variables" chapter), only the indexed bit is forced. The source variable and all other symbols that are indexed to the same variable, remain unforced. If you write to the source variable, the forced bit remains unchanged. If you force an indexed bit and then force the source variable, the second force overrides the first force. That is, when you unforce the variable, the bit is also unforced.
- If you add a variable that has been defined as a constant to the Watch window, the variable appears as a dimmed value. You cannot modify the variable in the Watch window.
- If you have added a variable to the Watch window and then you delete it from the Symbol Manager, a Smart Download or Smart Restart does not remove the variable from the Watch window or the runtime engine. In addition, the Watch window continues to allow you to modify the variable. Note that if your program is affected by the deletion of the symbol, it will fail to validate until you modify the logic appropriately.

  For more information about the behavior of variables at runtime, see the "Defining Variables" chapter.

Note The Administrator or Engineer security level is required in order to force or modify a variable.
A user's security privilege applies to the local node where the user is logged in and to any remote nodes to which the user can connect.

You have the option of backing up the value of a variable and its forced state to the hard disk. InControl provides three ways by which you can specify for the backup to occur.

- If the runtime engine shuts down during a power failure, the value of a forced variable is copied to the hard disk. Note that the values of any retentive variables are also saved during a power failure.

  The values of retentive and forced variables are not saved unless you are using an intelligent UPS with the system and you have configured it to signal InControl of the power failure .

  For more information about preparing for power failures, see "Handling Power Failure" in the "InControl System Administration" chapter.
- You can configure InControl to save retentive and forced variables to the hard disk periodically. The default frequency of zero disables this feature. You can change it in the **Runtime Engine Properties** dialog box, described in "Setting Scan Times" of the "InControl System Administration" chapter.
- You can design code in a program to save the value of retentive and forced variables on demand. For a forced variable, both the value and the forced state are saved to the hard disk. Use the following syntax:
  `RTEngine.ExecProjectCommand (SaveRetentive);`

The values are only restored when the runtime engine is configured to restart automatically (Last, Pause, Run mode) after a system reboot. For more information, see "Restarting Projects Automatically" in the "InControl System Administration" chapter.

You can write values to the Mode system variable for a program and thereby control the mode of that program from another program in the same project. The following table lists the valid values that can be written to the Mode system variables.

Valid Mode System Variable Values

Write This Value [1]	**This Action Occurs**
PAUSE (or 2)	Set program to Paused mode.
SCAN (or 3)	Single Scan a program. Program must be paused first.
RUN (or 5)	Set program to Run mode.
FAULT (or 6) [2]	Set program to Fault mode. Be sure that you are fully informed of the process under control by the program before setting the mode to Fault.
COMPLETE (or 7)	Sets the program mode to Complete. Functionally, the Complete and Paused modes are equivalent.

1 If you are monitoring the program Mode variable, the following additional values may be read, although they cannot be written to this variable:
Unknown (0) Program is unloaded from the runtime engine.
Stop (1) Program is stopped.
Loaded (4) Program is being loaded to the runtime engine.
Program (8) **For programs**:
The program is loaded to the runtime engine and is ready to run when called by another program. Applicable to functions, function blocks, and to FOEs that require function calls to a method in order to run. Note that the mode of these POUs does not change to Run, even after they are called.
Program (8) **For I/O configurations**:
The configuration has been downloaded to a remote node. Automatic configuration can be done on that remote node while the configuration is in the loaded mode.

2 For information about clearing a program fault, see "Checking the Program Mode."

If you force the Mode variable for a program, other InControl programs are blocked from writing new values to the variable. In addition, external programs, such as InTouch, or a Visual Basic application, are also blocked from writing new values to the variable. The commands on the toolbar and the **Runtime** menu override the Mode variable when it is forced. However, any external programs that are monitoring the variable read only the forced value, which may be incorrect if the mode has been changed from the toolbar or menu.

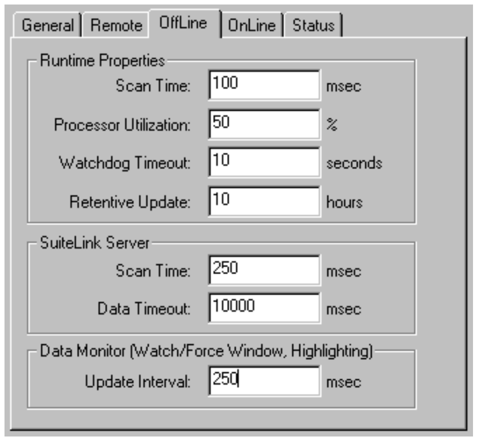
WARNING! When you modify mode variables in a program that uses the variables to control other programs, be sure that you are aware of the mode changes that take place in all programs. Unexpected program mode changes may cause injury or death and/or damage to equipment.

Adjusting Update Rate

Set the update rate for the Watch window in the **Runtime Engine Configuration** dialog box, online or offline.

To set the Watch window update rate:

1. Click **Configure** on the **Runtime** menu.
2. Select either the **Online** tab or the **Offline** tab. Changes made on the **Online** tab are retained for the current session only. Changes made on the **Offline** tab are saved.
3. Enter the update rate in the **Update Interval** field as shown in the following figure.



Runtime Properties

Scan Time:	100	msec
Processor Utilization:	50	%
Watchdog Timeout:	10	seconds
Retentive Update:	10	hours

SuiteLink Server

Scan Time:	250	msec
Data Timeout:	10000	msec

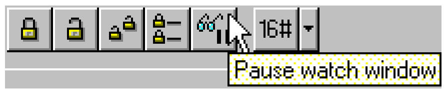
Data Monitor (Watch/Force Window, Highlighting)

Update Interval:	250	msec

Watch Window - Setting Update Rate

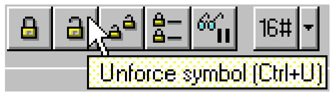
Pausing the Watch Window Update

You can pause the update for the Watch window by clicking **Pause Watch window** on the Watch window toolbar. The values are dimmed when the Watch window is paused.



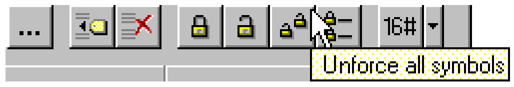
Unforcing Variables

To unforce a variable, click the name of the variable and click **Unforce Symbol**.



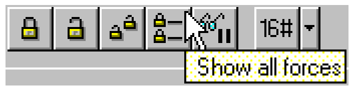
You can also unforce a variable from the **Modify Value** dialog box, described in "Modifying/Forcing a Variable."

To unforce all forced variables, including variables that have been forced in other tables, click **Unforce All Symbols**.



Displaying Forced Variables

To display all variables that have been forced, including variables that have been forced in other tables, click **Show All Forces**.



All variables in the project that have been forced are displayed at the top of the currently displayed table.

Note that displaying another table or exiting InControl causes the currently displayed table to be saved, including the list of forced variables.

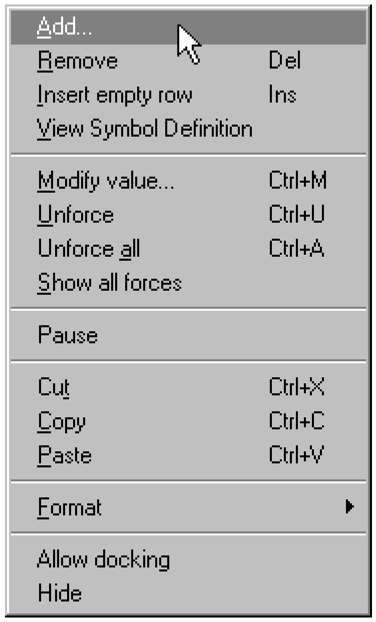
Using the Watch Window on a Remote Computer

You can open and edit a project on a computer when the runtime engine on the computer is running a different project. If you open the Watch window on this computer, the symbols that you can add to the Watch window by clicking **Add Symbol** are those defined for the project being edited, not the symbols in the project that is running. However, if you know the names of the symbols used in the project that is running, you can type them into the Watch window and monitor their status as the program runs.

As an alternative to typing in each symbol name within the Watch window, you can create an ASCII file, described in "Adding a Table of Variables to the Watch Window."

Using the Watch Window Menu

You can choose all the Watch window functions from a menu. Place the cursor within the Watch window and right-click. Click the appropriate function to select it.



Watch Window - Commands

Menu Bar	**Toolbar Icon**	**Description**
Add	➕	Add symbol to Watch window.
Remove	✗	Remove symbol from Watch window.
Insert Empty Row	n/a	Insert a row between variables.
View Symbol Definition	n/a	Display symbol definition in the Symbol Manager.
Modify Value	🔒	Write or force a value to selected variable.
Unforce	🔓	Unforce selected variable.
Unforce All	🔓	Unforce all variables that have been forced.
Show All Forces	🔒	Display all variables that have been forced.
Pause	⏸	Pause the update for the Watch window.
Cut	n/a	Cut selected item and place it on the clipboard.

Menu Bar	Toolbar Icon	Description
Copy	n/a	Copy selected item and place it on the clipboard.
Paste	n/a	Paste contents of clipboard.
Format	16#	Choose format (binary, octal, decimal, or hexadecimal) for ANY_INT data types (BYTEs, WORDs, DWORDs, etc.). If you change format while the Watch window is paused, the value currently displayed is reformatted. That is, a new value is not read from the runtime engine before reformatting is done.
Allow Docking [1]	n/a	Click to dock or undock the Watch window.
Hide [1]	n/a	Remove the Watch window.
1 Option does not appear on the menu when accessed from the stand-alone Watch window.		

The Watch window supports the standard cut/copy/paste features. This allows you to do the following:

- Duplicate or move symbols within the Watch window.
- Copy symbols from the Symbol Manager to the Watch window.
- Copy symbols from an external application, such as Excel, or Notepad.
- Copy symbols between multiple instances of the Watch window.

Using the Stand-Alone Watch Window

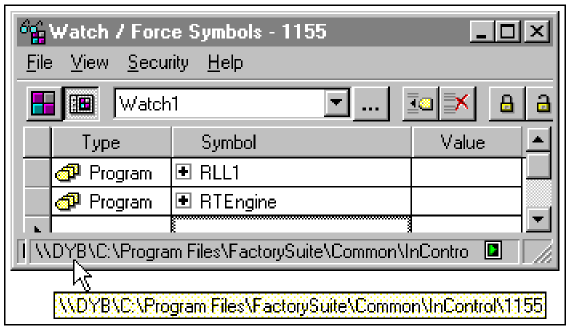
You can run a stand-alone version of the Watch window. The stand-alone Watch window operates very similarly to the Watch window that you open from the Development environment. However, with the stand-alone Watch window, you can monitor variables during runtime without opening the Development environment. In addition, you can run multiple instances of the stand-alone Watch window and monitor the variables on one or more nodes.

To run the standalone Watch window:

1. On the runtime engine monitor icon, click **Watch window**. You can also run it from the **Start** menu on the Task Bar.
2. If a dialog box prompts you for a node, enter the name of the node that you want to monitor. Depending on the target system, you may need to enter the TCP/IP address instead.

   For the local node, leave the **Node** field blank.

3. Verify the project that you intend to monitor. The node name and project appear in the Status bar at the bottom of the Watch window. If you place the cursor in the Status bar, the Tip help also displays the name of the project.



Type	Symbol	Value
Program	RLL1	
Program	RTEngine	

\\DYB\C:\Program Files\FactorySuite\Common\InContro

\\DYB\C:\Program Files\FactorySuite\Common\InControl\1155

Standalone Watch Window

The menu commands for the stand-alone Watch window are described in the following table.

Stand-Alone Watch Window Commands

Menu Bar Option	**Command**	**Toolbar Icon**	**Description**
File	Project	▦	Accesses Project Manager
	Connect / Disconnect	▣	Connects/disconnects the stand-alone Watch window and the Runtime environment.
	Save	n/a	Adds symbol to Watch window.
	Tables	...	Saves contents of table and displays list of watch tables.
	Add	▤	Adds symbol to Watch window.
	Remove	✗	Removes symbol from Watch window.
	Insert Empty Row	n/a	Inserts a row between variables.
	View Symbol Definition	n/a	Displays symbol definition in the Symbol Manager.
	Modify Value	🔒	Writes or forces a value to a variable.
	Unforce	🔓	Unforces selected variable.
	Unforce All	🔓	Unforces all variables that have been forced.
	Show All Forces	🔒	Displays all variables that have been forced.
	Cut	n/a	Cut selected item and place it on the clipboard.
File	Copy	n/a	Copy selected item and place it on the clipboard.
	Paste	n/a	Paste contents of clipboard.
	Exit	n/a	Closes Watch window.
View	Toolbar	n/a	Displays Watch window toolbar.
	Status Bar	n/a	Displays Watch window Status Bar.
	Format	16#	Choose format (binary, octal, decimal, or hexadecimal) for ANY_INT data types (BYTEs, WORDs, etc.) or ANY_REAL data types (REAL, LREAL).
	Pause	⏸	Pause the update for the Watch window.

Menu Bar Option	Command	Toolbar Icon	Description
Security [1]	Log On	n/a	Accesses **Log On** dialog box.
	Change Password	n/a	Accesses **Change Password** dialog box.
	Configure Users	n/a	Accesses **Configure Users** dialog box.
	Log Off	n/a	Accesses **Log Off** dialog box.
Help			Displays online help for the standalone Watch window.

1 Security configuration is described in the "Setting Up Security" chapter.

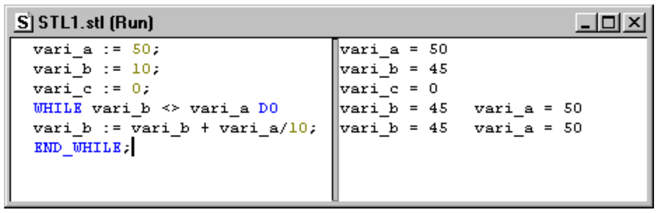
You can also choose functions for the stand-alone Watch window by placing the cursor within the Watch window and clicking the right mouse button. These functions are described in "Using the Watch Window Menu."

Note The Administrator or Engineer security level is required in order to force or modify a variable.
A user's security privilege applies to any remote nodes to which the user can connect, as well as to the local node where the user is logged in.

Using the Editor Window

For STL and SFC programs, you can monitor the values of variables from the editor window. When the program is running, the editor window displays global variables and variables that are local to the program.

- In an STL program, variables are displayed in the right half of the editor window when the program goes to the Run mode.
- In an SFC program, variables are displayed in the right half of the editor window for individual Steps. You must open each Step and display the STL code first.



```
vari_a := 50;                    vari_a = 50
vari_b := 10;                    vari_b = 45
vari_c := 0;                     vari_c = 0
WHILE vari_b <> vari_a DO        vari_b = 45   vari_a = 50
vari_b := vari_b + vari_a/10;    vari_b = 45   vari_a = 50
END_WHILE;
```

If program flow has stopped at a breakpoint in a function, the editor window shows the values of global variables and the values of variables that are local to the function. Variables local to the function are not displayed unless program flow has paused.

Checking the Wonderware Logger

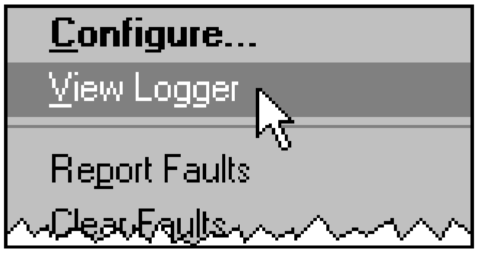
The Wonderware Logger records information regarding the activity done on the computer, i.e., startup data, error conditions, SuiteLink Server information, etc. The Wonderware Logger is a Windows service and is configured to start automatically when you power up the hardware unit.

If a problem occurs, for example, the runtime engine indicates an error condition, be sure to check the Wonderware Logger before clearing the error. Always check the Wonderware Logger for error messages before calling Technical Support.

The runtime engine monitor icon shown below indicates an error condition and that information has been written to the Wonderware Logger.

Note that information may be written to the Wonderware Logger even if no error condition occurs, for example mode changes.

- To display the Wonderware Logger, right-click the runtime engine monitor icon, and select **View Logger**, as shown below.



Accessing the Logger 2

For more information about the Wonderware Logger, see the *Log Viewer Online Help.*

The Windows Event Viewer also monitors events in your hardware unit. For more information about the Event Viewer, see your Windows documentation.

The Runtime Engine Logger contains information about the runtime engine when the hardware platform does not run the Windows operating system. For more information, see "Looking at Logger Data" in the "InControl System Administration" chapter."

Using the Runtime Engine System Variables

The InControl system variables provide additional information about the system, such as scan and program execution times, power failures, etc. See the Runtime Engine System Variables table in the Defining Variables chapter for a detailed description of these system variables.

CHAPTER 9

InControl System Administration

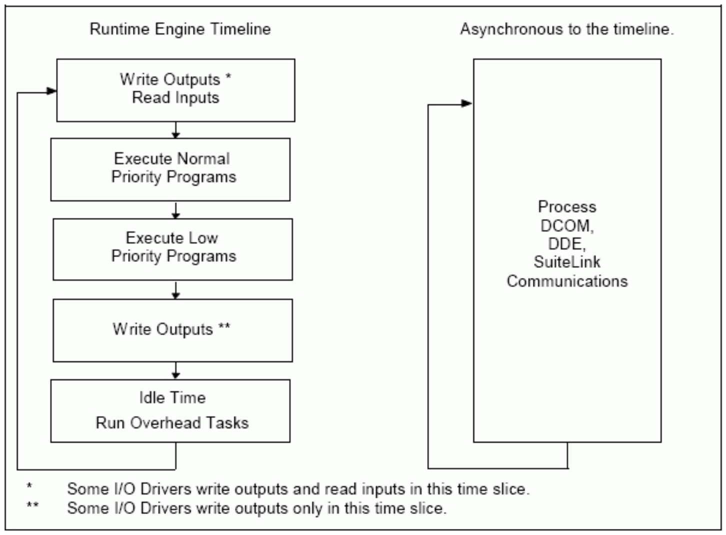
This chapter describes the InControl system-level functions.

Contents

Runtime Engine Timeline

The InControl runtime engine reads and writes to the I/O, executes application programs, and does overhead tasks and SuiteLink/DDE communications. The time required to do all these tasks, called the Total Scan Time, is adjustable on the **Runtime Engine Properties** dialog box.

A diagram of a complete scan by the runtime engine (the timeline) is shown in the following figure.


Runtime Engine Timeline
Asynchronous to the timeline.
Write Outputs *
Read Inputs
Execute Normal Priority Programs
Execute Low Priority Programs
Write Outputs **
Idle Time
Run Overhead Tasks
Process DCOM, DDE, SuiteLink Communications
* Some I/O Drivers write outputs and read inputs in this time slice.
** Some I/O Drivers write outputs only in this time slice.

InControl Timeline

Immediate reads and writes take place for global and local variables that are not I/O variables. You can use DCOM and SuiteLink interfaces to read and write intermediate values for RLL and SFC logic. This is done asynchronously, as shown in the figure. SuiteLink read/write operations from these interfaces occur immediately, whenever they are requested by the external application, e.g., InTouch. The frequency of a SuiteLink "Advise" is determined by the SuiteLink Server scan time. The frequency of a DCOM "Advise" is determined by the Data Monitor Update Interval. You specify both values in the **Runtime Engine Properties** dialog box.

Controller cards, control bus architectures, and I/O modules all have different timing requirements, which are often implementation specific. It is possible to set a total scan time within InControl that is faster than these devices can handle. For information about how to handle this situation, see "Setting Scan Times."

Because runtime engine execution has a high priority within the Windows operating system, it preempts other operations, such as mouse movement, for example. Decreasing the scan time could cause the system to respond more slowly to user inputs from the mouse or keyboard.

Details of the timeline are listed below. Program execution is described in the pages that follow.

1. Update the I/O for the first board listed in the I/O View. An update is based on the I/O driver's implementation; some boards write and read I/O in this time slice, while other boards only read inputs.

   Update additional boards in the same order in which they are listed in the I/O View.

2. Execute normal-priority programs in the order that they appear in the Execution View. * For projects, you specify the execution order in the Execution View, as described in "Project/Program Execution Order" in the "Running a Project" chapter. If you change the order in the Execution View, the new order does not take effect until the next project download.

   Normal-priority programs run in a real-time relationship (synchronous) with the I/O. The runtime engine always executes these programs. If the Total Scan Time is set too low, the programs are run, but the RTEngine.ScanOverrun system variable is set to TRUE.

3. Execute low-priority programs in the order that they are downloaded. * As with normal-priority programs, you can specify the download order in the Execution View. If a program is downloaded after a project is downloaded, its execution order as defined in the Execution View is preserved.

   Low-priority programs are run only if time remains after all the other tasks on the timeline have been executed. If all low-priority programs are not executed in a single scan, execution begins after the last low-priority program that ran during the previous scan.

   Note that low-priority programs execute as Windows high-priority real-time threads.

4. I/O boards that only read inputs in step 1 write outputs in this time slice.

5. Run overhead tasks. If the runtime engine begins to use more than the allocation percentage specified in the **Processor Utilization** field ("Setting Scan Times"), it begins to skip scans. This design prevents the runtime engine from consuming so much of the processor time that user interaction is blocked. If the scan does not complete before the watchdog timer, specified in the **Watchdog Timeout** field ("Setting Scan Times"), is reset, the runtime engine service is stopped.

   * When the runtime engine is in the Pause mode, normal- and low-priority programs are not executed.

SFC Execution

The general execution order for the program elements of an SFC is as follows:

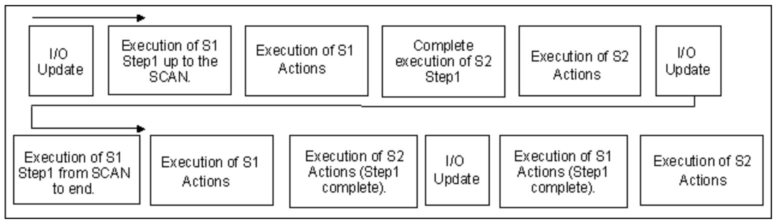
1. Evaluate all active Transitions. Actions and Steps are scheduled for execution.
2. Execute scheduled tasks: Step code (Structured Text or Macros).
3. Execute Actions.

The Structured Text code in a Step is executed one time to completion when the Step is first activated. Exceptions to this can occur in lengthy operations, such as loops (FOR, WHILE, REPEAT), and file access operations. For loops, execution of the Structured Text is suspended until the next scan, unless the loop is ended with one of the NOWAIT statements. The SCAN statement forces execution to suspend until after the next I/O scan.

The SFC does not transition until the Structured Text code is complete. In addition, after the code is complete, the Step does not run again until the Step is exited and reentered.

When an Action terminates, it is executed one more time on the following scan, with the rung input set to FALSE. This allows timers, counters, and output coils to reset. If you want additional logic to be executed when the Action is terminated, use the F_TRIG function block to negate the FALSE power flow into the rung.

In the following example, the project contains two SFCs, S1 and S2, each with one Step. If S1 has a SCAN statement halfway through its execution, the following order obtains:


I/O Update
Execution of S1 Step1 up to the SCAN.
Execution of S1 Actions
Complete execution of S2 Step1
Execution of S2 Actions
I/O Update
Execution of S1 Step1 from SCAN to end.
Execution of S1 Actions
Execution of S2 Actions (Step1 complete).
I/O Update
Execution of S1 Actions (Step1 complete).
Execution of S2 Actions

SFC Execution

Structured Text Program Execution

The execution of a Structured Text program consists of one complete scan of all statements in the program. Note that exceptions can occur in lengthy operations, such as loops (FOR, WHILE, REPEAT), and file access operations. For loops, execution of the Structured Text is suspended until the next scan, unless the loop is ended with one of the NOWAIT statements. The SCAN statement forces execution to suspend until after the next I/O scan.

RLL Execution

The execution of a Relay Ladder Logic program consists of one complete scan of the program.

FOE Execution

The execution of an FOE consists of calling its control method once per scan. For ActiveX controls that are not InControl specific, an SFC or Structured Text instruction must call the ActiveX control whenever the control is to be run.

Some FOEs, such as the PID FOE, are designed to be executed automatically. They have a method that is called once every scan by the runtime engine. When loaded to the runtime engine, the mode for these FOEs can be Run, Pause, Stop, etc.

Some FOEs do not have a method designed to run automatically every scan. For these FOEs, you need to include code in an SFC or Structured Text program to call a method for the FOE to execute correctly. When loaded to the runtime engine, the mode for these FOEs is indicated as being Loaded.

Program Execution and Stepping a Program

The Step Program option allows you to execute a line of logic in a single program. A program must be in the Pause mode before you can step it.

Each time you click **Step Program** on the Runtime toolbar, programs are executed as follows:

- For an RLL program, one rung of logic is executed.
- For a Structured Text program, one line of code is executed.
- For an SFC program, one line of code of an active Step is executed. If more than one Step is active, each Step is stepped through in turn. All active Actions are executed completely.
- For an FOE, the Step Program option is not supported.

Note Single-stepping any program may cause execution order to differ from that of a normally running program. This is due to the continuous execution of I/O, other programs, and in the case of SFCs, Actions.

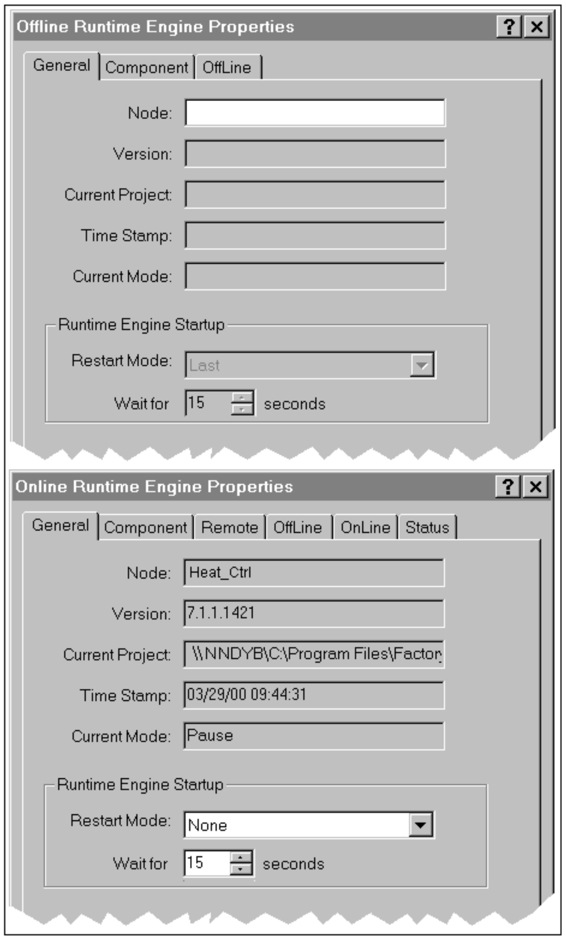
Project/Program Execution and Single Scanning

The Single Scan option allows you to execute all programs in a project for one complete scan, following the timeline that is described in "Runtime Engine Timeline." Order of program execution is based on the order of the programs in the Execution View. A project/program must be in the Pause mode before you can execute a single scan.

Note Single-scanning may cause program execution order to differ from that of normally running programs. This is due to the continuous execution of I/O, other programs, and in the case of SFCs, Actions.

Accessing the Runtime Engine Properties

To configure the runtime engine, click **Configure** on the **Runtime** menu. A different dialog box appears, depending on whether the development environment is connected to the runtime engine. If the development environment is not connected, the **Offline Runtime Engine Properties** dialog box appears. If the development environment is connected, the **Online Runtime Engine Properties** dialog box appears. Additional tabs appear in the **Online Runtime Engine Properties** dialog box, as shown below.



Offline Runtime Engine Properties — General

Node:	
Version:	
Current Project:	
Time Stamp:	
Current Mode:	
Restart Mode:	Last
Wait for	15 seconds

Online Runtime Engine Properties — General

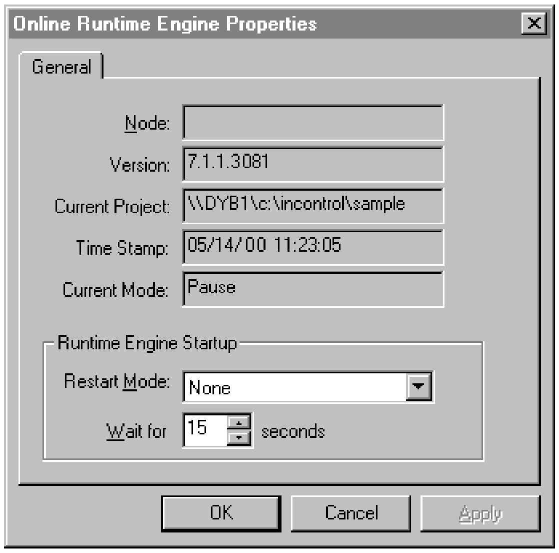
Node:	Heat_Ctrl
Version:	7.1.1.1421
Current Project:	\\NNDYB\C:\Program Files\Factor
Time Stamp:	03/29/00 09:44:31
Current Mode:	Pause
Restart Mode:	None
Wait for	15 seconds

Accessing Runtime Engine Properties

The property sheets for the runtime engine are described in the following pages.

Checking General Properties of the Runtime Engine

The General tab of the **Runtime Engine Properties** dialog box is shown in the following figure.



Node:
Version: 7.1.1.3081
Current Project: \\DYB1\c:\incontrol\sample
Time Stamp: 05/14/00 11:23:05
Current Mode: Pause
Runtime Engine Startup
Restart Mode: None
Wait for 15 seconds

Runtime Engine Properties-General Tab

Configuration may vary for different target hardware platforms. For more information about configuring the runtime engine on another platform, see the appropriate user's guide for that hardware platform.

Runtime Engine Properties-General Tab

Field	Description
Node	**Connected to runtime engine** Read-only field displays node to which the Development environment is connected. **Not connected to runtime engine** Enter the name of the node where you want to run the project. For the local node, leave the field blank.
Version	**Connected to runtime engine** Read-only field displays version of the runtime engine. **Not connected to runtime engine** Field is blank.
Current Project	Connected to runtime engine Read-only field displays project downloaded to runtime engine. Not connected to runtime engine Field is blank.
Time Stamp	Read-only field displays last time any changes were made to the project downloaded to the runtime engine. Field is blank when not connected to the runtime engine, or if no project is downloaded to the runtime engine.

Field	Description
Current Mode [1]	**Connected to runtime engine** Read-only field displays the current status of the runtime engine: Run, Pause, Stop. These modes also occur, but are unlikely to be seen: Fault—The runtime engine enters the Fault mode. Note that this is different from a program fault. Single Scan—A single scan is being run. Program—A validation and download is taking place. **Not connected to runtime engine** Field is blank.
Restart Mode [2]	**Connected to runtime engine** Enter mode that the runtime engine automatically enters after the hardware unit is booted. Last—Reload last project to the runtime engine and set it to mode that it was in before hardware unit was booted. If you do a manual shutdown, compared to a shutdown due to power failure, InControl sets the runtime engine to Stop. Therefore, when you reboot, the runtime engine enters the Stop mode. None—The runtime engine service starts, but no programs are loaded in runtime engine. Pause— Reload last project to the runtime engine and set it to the Pause mode. Run— Reload last project to the runtime engine and set it to the Run mode. **Not connected to runtime engine** Read-only field displays default values.
Wait For	**Connected to runtime engine** Enter zero for the project to resume running as soon as possible after the runtime engine service restarts. Enter a delay in seconds before a project automatically resumes running after the runtime engine service restarts. This allows you time to cancel if necessary. **Not connected to runtime engine** Read-only field displays default values.

1 Mode is also indicated by the RTEngine.Mode system variable, described in "Using the Runtime Engine System Variables" of the "Running a Project" chapter.
2 For more about automatic startup, see "Restarting Projects Automatically."

Setting Scan Times

Use the Offline or Online tabs of the **Runtime Engine Properties** dialog box to designate the processor utilization and to set timing intervals, such as scan times, and update time for the Watch Window.

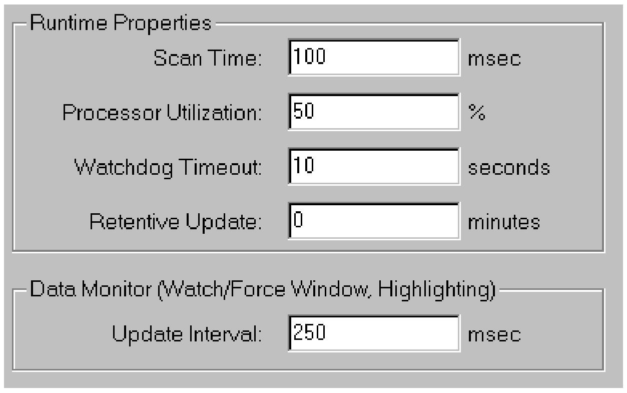
- Offline tab — Changes that you make on the Offline tab take effect the next time you download a project or individual program. The Offline tab appears when you click **Configure** on the **Runtime** menu or double-click the **RTEngine** icon in the Project View.
- Online tab — Changes that you make on the Online tab take effect as soon as you apply them. However, the next time you load a project or program and include the project configuration (select **Full Restart** on the **Run Project** dialog box, for example) the settings of the Offline tab overwrite the settings of the Online tab.

  The Online tab appears when you place the cursor over the runtime engine monitor icon in the task bar, right-click, and click **Configure**. If the development environment is connected to the runtime engine, click **Configure** on the **Runtime** menu or double-click the **RTEngine** icon in the Project View.

WARNING! Setting the Scan Time too low may cause the runtime engine to skip scans. This could result in unpredictable operation by the I/O devices and cause injury or death and/or damage to equipment. Be sure to select an appropriate scan time for your application. See "Setting Scan Times" for more information.

Note Configuration may vary for different target hardware platforms. For more information about configuring the runtime engine on another platform, see the appropriate user's guide for that hardware platform.

The fields on the Offline and Online tabs of the **Runtime Engine Properties** dialog box are shown in the following figure.


Runtime Properties
Scan Time: 100 msec
Processor Utilization: 50 %
Watchdog Timeout: 10 seconds
Retentive Update: 0 minutes
Data Monitor (Watch/Force Window, Highlighting)
Update Interval: 250 msec

Runtime Engine Properties-Offline/Online Tab

Field	Description
Scan Time / Processor Utilization	Enter values in the **Scan Time** and **Processor Utilization** fields that act together to determine how much processor time the runtime engine has to execute logic and process I/O. Scan time determines how often the runtime engine executes logic. Processor utilization determines the maximum percent of the scan time that can be used by the runtime engine in the execution of program logic and processing I/O. Changes that you make on the Offline tab take effect the next time you download a project or individual program. For more information about setting the scan, see "Setting Scan Times."
Watchdog Timeout	Specify the time interval (greater than the scan time) during which the runtime engine must reset the watchdog timer. The runtime engine service is stopped if the watchdog timer is not reset during this interval. It is recommended that you reboot the system to restart the runtime engine service. If the watchdog timer expired because a program entered an endless loop, cancel the automatic restart if the runtime engine is configured to run the project again. Changes that you make on the Offline tab take effect the next time you download a project or individual program.
Retentive Update	Specify the time interval at which retentive and forced values are copied to the hard disk. Fractional minutes are allowed. Enter 0 to disable. Take into account the number of retentive variables in your project when you choose the frequency of the update. A short update interval can degrade the performance of your system when a large number of the variables are retentive. Changes that you make on the Offline tab take effect the next time you download a project or individual program.
Update Interval	Specify the frequency for updating the Watch Window and runtime animation for those programs that support animation. Changes that you make on the Offline tab take effect the next time you download a project or individual program.

Tuning the Scan

It is important to tune the runtime engine to balance its CPU requirements with those of other applications, including the operating system. This will help avoid consuming all the resources of the CPU, which could lock up the system.

Targeting CPU Utilization

InControl provides two properties on the **Runtime Engine Properties** dialog box. You use these properties to determine the amount of processor time that the runtime engine has to execute logic and to process I/O.

- The Scan Time property determines how often the runtime engine executes logic.
- The Processor Utilization property determines the maximum percent of the scan time that can be used by the runtime engine in the execution of program logic and processing I/O.

You must specify appropriate values for these two parameters to balance the needs of the runtime engine with the needs of any other applications that are running.

On each scan, the runtime engine makes the following computation to determine a CPU percentage:

$$\text{CPU\%} = \frac{\text{Execution Time(Nomal Priority Program) + I/O Time}}{\text{Scan Time}}$$

Note The CPU percentage is the same for a single processor- or a multiple-processor machine. CPU percentage is based on time taken to execute, not on actual amount of CPU Utilization.

When the runtime engine does these computations, it checks to see if the CPU percentage exceeds the value that you have specified. If your value is exceeded, then the runtime engine will skip scans to bring the observed CPU percentage back to your value.

For example, if you specify 20% CPU Utilization and a 100 ms scan rate, then 20 ms will be allocated on each scan for normal-priority program execution and I/O scan. If the programs and I/O consistently take 30 ms, then the runtime engine will start skipping every eighth scan to bring the CPU percentage back to 20%. If the normal-priority programs and I/O consistently take 80 ms, then the runtime engine will run one scan and then skip three scans to bring the CPU percentage back to 20%. In both of these cases, if you specify 80% for the CPU Utilization, no scans are skipped.

Consider the runtime engine to have a target percentage of CPU use, and it will do the necessary calculations and skip the necessary scans to reach that target.

WARNING! Skipping scans could result in unpredictable operation by the I/O devices and cause injury or death to personnel and/or damage to equipment. To help avoid skipping scans, be sure to select a scan time and CPU utilization that is appropriate for your application.

Normally, some fluctuation occurs in both program execution time and I/O execution time from scan to scan. Therefore, it is recommended that you leave some additional room in the CPU percentage computation to accommodate these normal fluctuations.

Using the scan time and CPU percentage to throttle the runtime engine has the following goals:

- Permit other applications (with some limitations) to run on the same hardware unit as the runtime engine.
- Prevent the system from locking up in case of unintended overuse of the CPU by the program logic or I/O scan.

The mechanism for this throttling is designed to help in situations where programs and I/O are not operating normally. Skipping scans should not occur during normal operation.

Low-priority programs are treated in a different fashion. If the runtime engine has not used up its allotted time slice, then a low-priority program will be run. This low-priority program can run beyond the CPU percentage boundary, and it will not be counted as part of the runtime engine CPU percentage. If the first low-priority program finishes executing and some time remains in the runtime engine time slice during that scan, then another low-priority program will be started.

Note On a single-processor system, it is recommended that you avoid setting CPU Utilization to 100%. This is to allow time for the Windows operating system tasks to execute. In general, depending on your system and the applications that you are running, do not set CPU Utilization greater than 90%. For a multiple-processor system, a CPU Utilization of 100% is recommended.

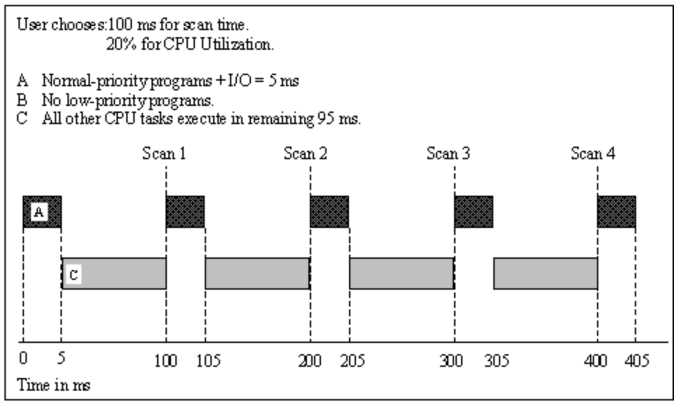
The figures on the next several pages illustrate various scenarios of program execution and show examples of normal operation and operation with skipped scans.

Examples of Normal / Skipped Scans

In the figure below, the ratio of actual CPU% to specified CPU Utilization is defined by the following relationship:

0 < ratio < 1

- Operation is normal.
- No scans are skipped.
- There are no watchdog errors.
- There are no low-priority programs.

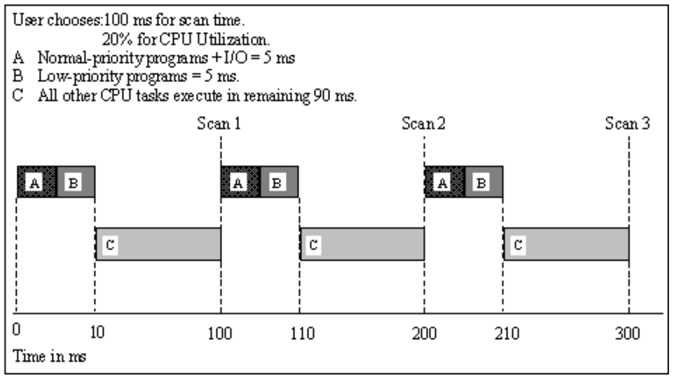

User chooses:100 ms for scan time.
20% for CPU Utilization.
A Normal-priority programs + I/O = 5 ms
B No low-priority programs.
C All other CPU tasks execute in remaining 95 ms.
Scan 1
Scan 2
Scan 3
Scan 4
A
C
0
5
100
105
200
205
300
305
400
405
Time in ms

InControl Scans Example 1

In the figure below, the ratio of actual CPU% to specified CPU Utilization is defined by the following relationship:

0 < ratio < 1

- Operation is normal.
- No scans are skipped.
- There are no watchdog errors.
- Low-priority programs are scheduled to run.

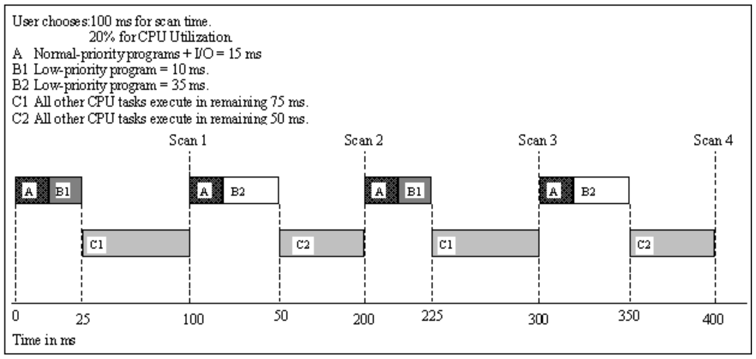

User chooses:100 ms for scan time.
20% for CPU Utilization.
A Normal-priority programs + I/O = 5 ms
B Low-priority programs = 5 ms.
C All other CPU tasks execute in remaining 90 ms.
Scan 1
Scan 2
Scan 3
A
B
C
0
10
100
110
200
210
300
Time in ms

InControl Scans Example 2

In the figure below, the ratio of actual CPU% to specified CPU Utilization is defined by the following relationship:

0 < ratio < 1

- Operation is normal.
- No scans are skipped.
- There are no watchdog errors.
- Low-priority programs are scheduled to run and they extend the scan.

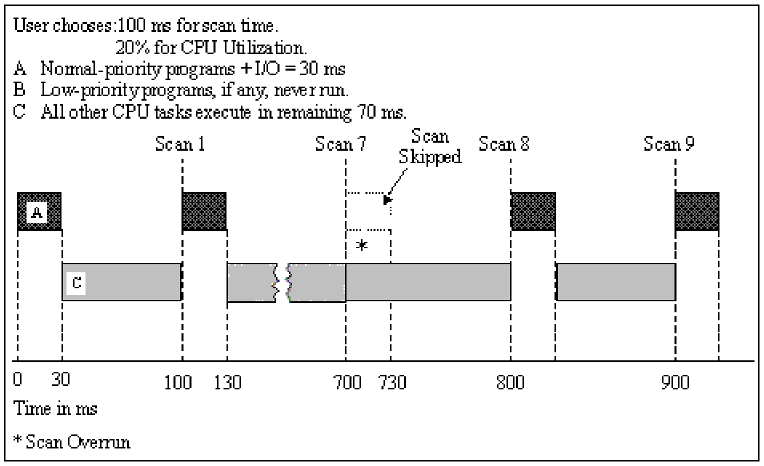

User chooses:100 ms for scan time.
20% for CPU Utilization.
A Normal-priority programs + I/O = 15 ms
B1 Low-priority program = 10 ms.
B2 Low-priority program = 35 ms.
C1 All other CPU tasks execute in remaining 75 ms.
C2 All other CPU tasks execute in remaining 50 ms.
Scan 1
Scan 2
Scan 3
Scan 4
A
B1
B2
C1
C2
0
25
100
50
200
225
300
350
400
Time in ms

InControl Scans Example 3

In the figure below, the ratio of actual CPU% to specified CPU Utilization is defined by the following relationship:

1 < ratio < 2

- Operation is not normal.
- Scans are skipped.
- There are no watchdog errors.
- Low-priority programs, if any, will never run.

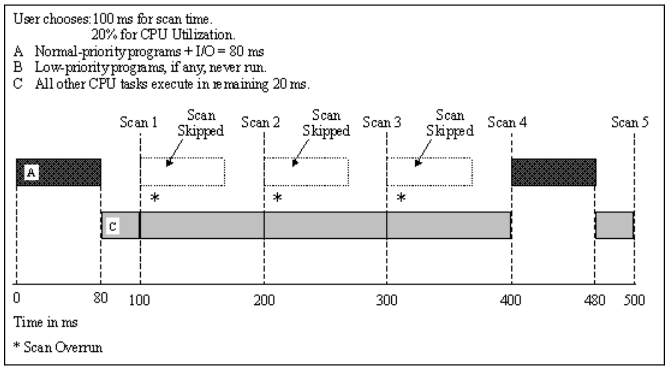

User chooses:100 ms for scan time.
20% for CPU Utilization.
A Normal-priority programs + I/O = 30 ms
B Low-priority programs, if any, never run.
C All other CPU tasks execute in remaining 70 ms.
Scan 1
Scan 7
Scan Skipped
Scan 8
Scan 9
A
C
*
0
30
100
130
700
730
800
900
Time in ms
* Scan Overrun

InControl Scans Example 4

In the figure below, the ratio of actual CPU% to specified CPU Utilization is defined by the following relationship:

ratio => 2

- Operation is not normal.
- Scans are skipped more aggressively.
- There are no watchdog errors.
- Low-priority programs, if any, will never run.


User chooses: 100 ms for scan time.
20% for CPU Utilization.
A Normal-priority programs + I/O = 80 ms
B Low-priority programs, if any, never run.
C All other CPU tasks execute in remaining 20 ms.
Scan 1
Scan Skipped
Scan 2
Scan Skipped
Scan 3
Scan Skipped
Scan 4
Scan 5
A
C
*
0
80
100
200
300
400
480
500
Time in ms
* Scan Overrun

InControl Scans Example 5

Scan Operation and the Watchdog Timer

A watchdog error occurs when a single scan of program logic and I/O exceed the specified watchdog interval. The runtime engine makes this check at every scan interval. If the watchdog timer ever expires, then the runtime engine attempts to set the I/O to a safe state and then shuts down. The default watchdog interval is 10 seconds.

I/O Considerations

Controller cards, control bus architectures, and I/O modules all have different timing requirements, which are often implementation specific. It is possible to set a total scan time within InControl that is faster than these devices can handle. This is particularly true when the scan time is less than 10 ms. For example, you can set InControl to update a 300 Hz I/O module at 1000 Hz, but only 300 of these updates actually register with the I/O module.

For scan times under 10 ms, it is recommended that you do initial configuration and testing at a scan time over 10 ms. Then test for successful performance at progressively faster scan times, until the required scan time is achieved.

Check the actual scan times in the Status tab of the **Runtime Engine Properties** dialog box. Reduce the total scan time and CPU percentage until the average value is as low as necessary. Scan time is set relative to the amount of CPU time you use in your process. The relationship for determining the appropriate scan time is given in "Targeting CPU Utilization."

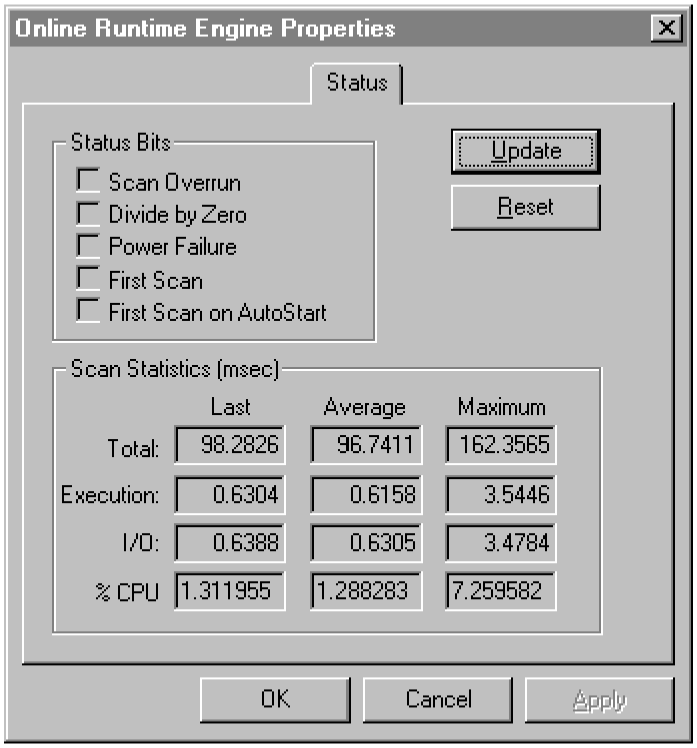
At very fast scan times (less than 10 ms) the runtime engine takes priority over the system call to the system performance timer. Consequently, the RTEngine.ScanLast and RTEngine.ScanMax system variables are not accurate reflections of actual I/O scan times at very fast scan rates. In addition, bus and I/O latencies for I/O modules can be in the range of 1-10 ms.

When your application requires scans of this speed, always connect an oscilloscope to the I/O to obtain accurate measurements of performance, instead of relying on the system variables for this data. InControl counters reflect only internal time.

WARNING! Changing scan time while programs are running may cause unpredictable operation by I/O devices, resulting in death or injury to personnel and /or damage to equipment. It is recommended that you set the runtime engine to Stop mode before changing the scan time. Then restart the programs.

Checking Runtime Status Data

The Status tab of the **Runtime Engine Properties** dialog box is shown in the following figure. Use this property sheet to check or reset scan times, status bits, and usage of memory allocated for InControl.



	Last	Average	Maximum
Total:	98.2826	96.7411	162.3565
Execution:	0.6304	0.6158	3.5446
I/O:	0.6388	0.6305	3.4784
% CPU	1.311955	1.288283	7.259582

Status of Runtime Engine Tab

Field	Description
Scan Overrun	Indicates when a scan overrun occurs.
Divide by Zero	Indicates when a division by zero occurs.
Power Failure	Indicates a power failure has occurred. In order to implement an uninterruptible power capability for your InControl system, you need to install an intelligent UPS. For more information, see "Handling Power Failure."
First Scan	Indicates that the current scan is the first scan. Useful for detecting the first scan within program logic.
First Scan on AutoStart	Indicates that the current scan is the first scan after an automatic restart.
Total	Amount of time to do a complete scan, including I/O scan, program execution, and idle time. If the time for a complete scan (value in the **Total/Last** field) is greater than the configured scan (value entered in Online tab), scans are skipped.
Execution	Amount of time to execute all programs currently running in the runtime engine.

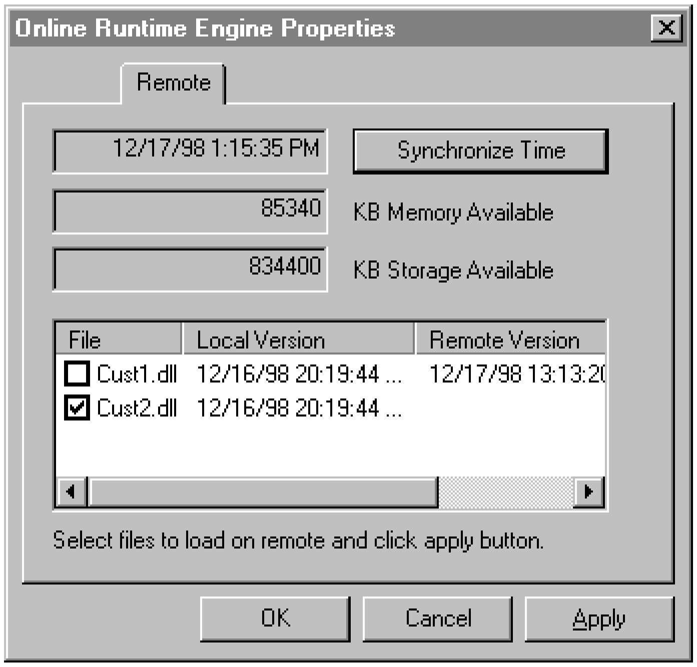
Field	Description
I/O	Amount of time to scan the I/O.
% CPU	Percent of processor time used by InControl. Used to monitor CPU percentage.
Update	Updates the values in the fields.
Reset	Sets all values displayed in the dialog box to an initial value. Scan statistics are set to zero. Status bits are set to FALSE.

Checking the Remote Node

The Remote tab appears in the **Online Runtime Engine Properties** dialog box when the target runtime engine is on a remote node.

Using the Remote Tab

The Remote tab of the **Runtime Engine Properties** dialog box is shown in the following figure.



12/17/98 1:15:35 PM

85340 KB Memory Available

834400 KB Storage Available

File	Local Version	Remote Version
☐ Cust1.dll	12/16/98 20:19:44 ...	12/17/98 13:13:2
☑ Cust2.dll	12/16/98 20:19:44 ...	

Select files to load on remote and click apply button.

Remote Node Tab

Field	**Description**
Time Field	Displays current time on the remote node. If the platform does not support daylight savings time, the value in this field is automatically corrected.
Synchronize Time Button	Click to set the time on the remote node to match the current time, daylight savings option, and time zone of the local node.
Memory Available	Displays the amount of physical memory (RAM) available on the remote node. Field is updated only when you open the dialog box.
Storage Available	Displays the amount of hard disk space available on the hard disk of the remote node where the runtime engine is installed. Field is updated only when you open the dialog box.

Field	Description
File Column	Lists files on the local node available for downloading to the remote node.
Local Version Column / Remote Version Column	Displays time stamp and version (where appropriate) for files that are available for downloading to the remote node, or that have already been downloaded. If **Remote Version** is blank, the file does not exist on the remote node.

Downloading Files

In the event that you need to download files to the remote node from the local node, you can use the Remote tab to check file versions and to select the files to be downloaded. For example, you may have a custom-designed factory object that you want to use on the remote node; or you may want to update an I/O driver on the remote node.

Before you can download any files, place copies of the files into a subdirectory of the InControl directory of the local node. This subdirectory must have the same name as the target name of the remote node. For example, if the path for InControl on the local system is the following:

```
C:\Program Files\FactorySuite\Common\InControl
```

and the target is a Windows NT platform, place the files in the following directory:

```
C:\Program Files\FactorySuite\Common\InControl\NT
```

If the target system is a Contec platform, for example, place the files in the following directory:

```
C:\Program Files\FactorySuite\Common\InControl\Contec
```

After you copy a file into the appropriate directory, the file name appears in the **File** field of the Remote tab. Select the files that you want to download and click **Apply**. Executable files, with the extensions .OCX, .DLL, and .EXE, are automatically registered on the remote node when they are downloaded.

A file cannot be transferred if it is in use. For program files that are running in the runtime engine on the remote node, you must stop the program before you can transfer it. Depending on the program and such factors as the length of its time-out period, you may have to wait several minutes for the program to be unloaded.

Note An error message may appear, depending on the file being downloaded, the target operating system and the way this operating system normally handles file registration. It is likely that the download and registration occurred correctly; however, be sure to check the appropriate user documentation for the target operating system for more information.

Synchronizing Time

Use the Remote tab to synchronize time on the remote node with that of the local node. Click **Synchronize Time** on the Remote tab to set the time on the remote node to match the current time, daylight savings option, and time zone of the local node.

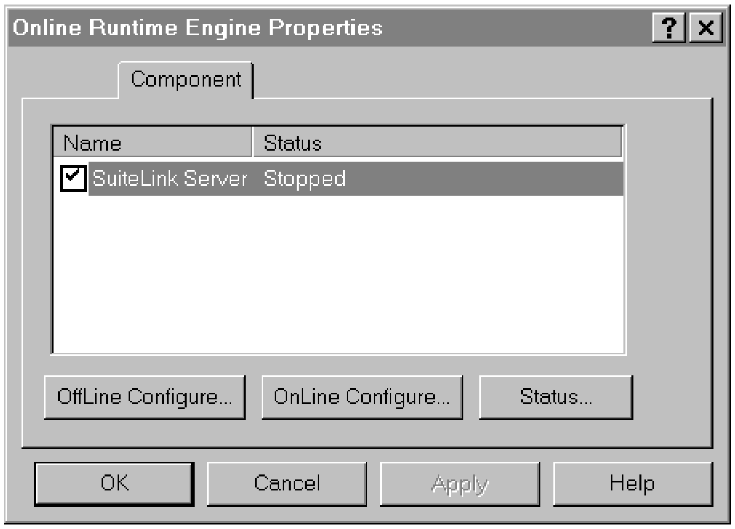
Note Some platforms do not support daylight savings time.

Configuring Components

The Components tab displays any new features (components) that you have installed your system to enhance the capabilities of the runtime engine. For example, if you install a third-party HMI, it is listed on the Components tab.

Using the Components Tab

The Components tab of the **Runtime Engine Properties** dialog box is shown in the following figure.



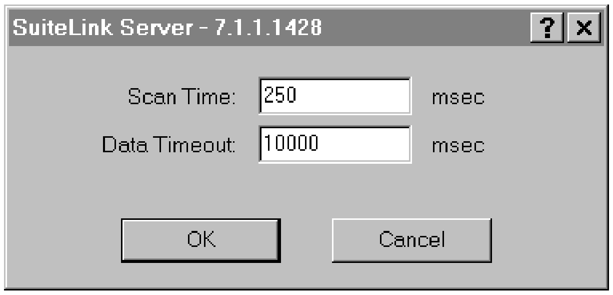
Name	Status
✓ SuiteLink Server	Stopped

Components Tab

Button / Field	Description
Name Column	Displays the name of the component. Select the checkbox to load the component; clear the checkbox to unload the component.
Status Column	Displays the current status of the component. Start = The component is loaded and running. Stop = The component is loaded but not running. Pause = The component is loaded but has temporarily stopped running. Disabled = The component is disabled. Not installed = The component is not installed.
Offline Configure	Click to set runtime parameters. Offline changes take effect after you download the project to the runtime engine.
Online Configure	Click to set runtime parameters. Online changes take effect as soon as you apply them.
Status	Click to change the status of the component.

SuiteLink Component Configuration

The SuiteLink Server is an example of a component that you may have installed and intend to run on your system. If you click **Offline Configure** or **Online Configure** on the Component tab, you can set the scan time and data timeout as shown in the following figure.


SuiteLink Server - 7.1.1.1428

Scan Time: 250 msec

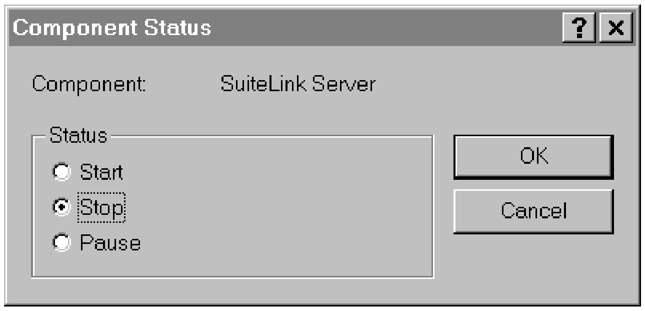
Data Timeout: 10000 msec

SuiteLink Configuration

Field	Description
Scan Time	Specify the time interval at which the runtime engine updates SuiteLink processes, such as InTouch or InSQL.
Data Timeout	Specify the time interval during which the SuiteLink server must respond to communications from the runtime engine before a timeout occurs.

SuiteLink Component Status

Click **Status** on the Component tab to change the component status as shown in the following figure.


Component Status

Component: SuiteLink Server

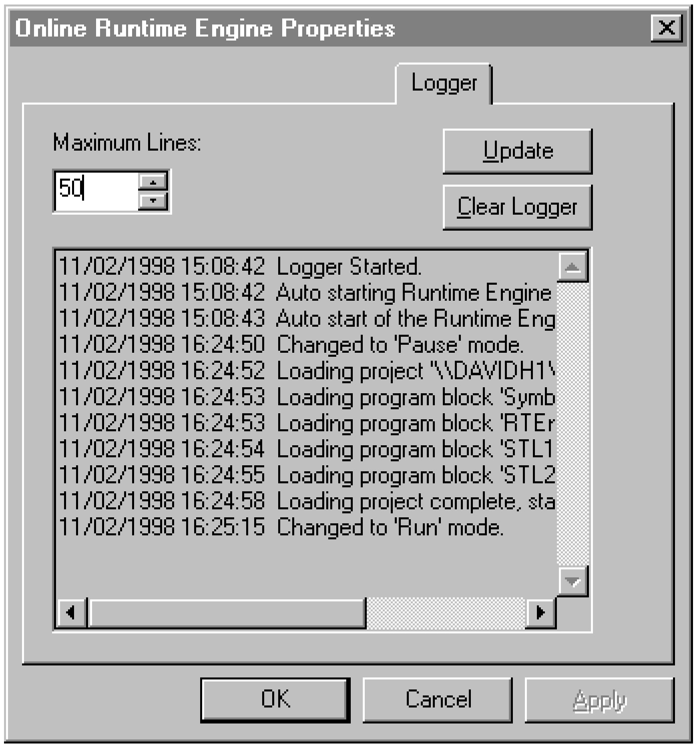
Status: ○ Start [✓ Stop] ○ Pause

SuiteLink Component Status

Button	Description
Start	Select to load and run the component.
Stop	Select to stop execution of the component.
Pause	Select to pause execution of the loaded component.

Looking at Logger Data

The Logger tab of the **Runtime Engine Properties** dialog box is shown in the following figure. This tab appears in the **Online Runtime Engine Properties** dialog box when the target runtime engine is on a hardware platform that is not using the Windows NT / 2000 operating system. When the hardwareplatform does not support the Wonderware Logger, all information that the runtime engine normally sends to the Wonderware Logger appears in the Runtime Engine Logger instead.



Maximum Lines: 50

```
11/02/1998 15:08:42  Logger Started.
11/02/1998 15:08:42  Auto starting Runtime Engine
11/02/1998 15:08:43  Auto start of the Runtime Eng
11/02/1998 16:24:50  Changed to 'Pause' mode.
11/02/1998 16:24:52  Loading project '\\DAVIDH1\
11/02/1998 16:24:53  Loading program block 'Symb
11/02/1998 16:24:53  Loading program block 'RTEr
11/02/1998 16:24:54  Loading program block 'STL1
11/02/1998 16:24:55  Loading program block 'STL2
11/02/1998 16:24:58  Loading project complete, sta
11/02/1998 16:25:15  Changed to 'Run' mode.
```

Logger Tab

Field / Button	Description
Maximum Lines	Enter the number of messages to display. When this number is reached, the oldest line is deleted to make room for a new message. To disable the Runtime Engine Logger, enter zero.
Update	Click to update the Logger with the latest values from the runtime engine.
Clear Logger	Click to clear all messages from the Logger. Note that this deletes messages permanently.

Clearing Runtime Engine Fault Mode

The runtime engine monitor checks the status of the runtime engine every 10 seconds. If the monitor is unable to obtain the status of the engine, the fault symbol is superimposed over the runtime engine monitor icon. In the event that the runtime engine enters the Fault mode, the system variable RTEngine.Mode is set to a value of six, and the following fault symbol is superimposed over the runtime

⊘

To clear the runtime engine Fault mode:

1. Check the Wonderware Logger or the Output window for error messages.
2. Click the runtime engine monitor icon and select **Exit Monitor**.
3. Restart the runtime engine service.
4. Redownload the project.

If a warning or error message is sent to the Wonderware Logger, the system variable RTEngine.Error is set to TRUE.

The RLL ABTAL function block and the Structured Text ABORT_ALL procedure can also set the runtime engine to the Fault mode. To clear the Fault mode in this case, you can skip steps 2 and 3 in the procedure above. Instead of step 4, set the runtime engine to the Pause or Run mode.

Clearing Program Fault Mode

During the course of its execution, a program may do an illegal operation and enter the Fault mode. The InControl validation process helps you avoid some programming mistakes, such as syntax errors, that result in illegal operations. However, some problems may still occur and cause a program to enter the Fault mode. For example, an array index may be assigned a value that is out of bounds. A third-party FOE may develop an internal error. A value of six may be written to a program's Mode system variable (from the Watch window, by program logic, etc.) and cause the program to enter the Fault mode.

If a program enters the Fault mode, "Fault" appears in the program's title bar and in the Project View. InControl sets the Mode system variable for the program to six, writes a message to the Output window and the Wonderware Logger, and

When a Program is in the Fault mode, its logic is no longer executing. A Faulted program does not directly affect the execution of other programs or POUs of the project

To restart a program in the Fault mode:

1. Clear faults. On the **Runtime Engine Monitor** menu, click **Clear Faults**. This sets the program to the Pause mode. If other programs are running, they continue to run and the runtime engine remains in the Run mode.

   You can also set the program to the Run or Pause mode to clear faults.
2. Set the program to the Run mode.
3. If the program returns to the Fault mode, you need to check program logic and possibly the integrity of your hardware. If you change the program, download it again and run it.

For information about using one of the runtime engine functions to clear faults, see "Checking the Program Mode" in the "System Administration" chapter.

Handling I/O and Other Hardware Errors

For I/O and other hardware errors, correct the fault condition and then click **Clear Faults** on the **Runtime** menu.

Note Information may be written to the Wonderware Logger even if no error condition occurs, for example when mode changes occur.

Configuring Runtime Engine Service Startup

The runtime engine is a Windows service and is configured to start automatically when you power up the hardware unit. Typically, it is not necessary to stop the runtime engine service. If no projects are running, the runtime engine is idle and consumes very little processor time. However, it may become necessary to stop the runtime engine service to install software, troubleshoot system problems, power cycle the system without automatically restarting the runtime engine etc. You can configure your system so that the runtime engine does not start automatically after you reboot the Windows operating system.

WARNING! Some system services are critical to the operation of the Windows operating system. If you stop an operating system service, the system may lock up and become difficult to reboot. This can result in unpredictable operation by I/O devices, which can cause death or injury to personnel and/or damage to equipment. Be sure to allow only qualified personnel to start and stop processes in the Services dialog box.

Handling Power Failure

You can configure InControl to shut down and then restart automatically in an orderly manner. This allows you to design an unattended resumption of control of your factory process in the event of a power failure.

Using an Uninterruptible Power Supply

InControl provides two utilities (ICPwFail and ICPwOn) and a system variable (RTEngine.PowerFail) for reporting the loss and restoration of power to the system. These features are designed to interact with any third-party uninterruptible power supply (UPS) that is capable of reporting a loss of power.

- The utility ICPwFail sets RTEngine.PowerFail to TRUE when ICPwFail receives notification from the UPS that power has been lost.
- The utility ICPwOn sets RTEngine.PowerFail to FALSE when ICPwOn receives notification from the UPS that power has resumed.

It is important to note that it is not sufficient simply to connect the InControl system to a UPS to monitor power status within your application. The UPS must have a programming capability that can be used to instruct it to respond to changes in power status in the following ways.

- When the UPS software determines that power has failed, it executes the ICPwFail utility. The utility communicates with InControl, permitting any actions that you have programmed to occur.
- When the UPS software determines that power has been restored, it executes the ICPwOn utility. The utility communicates this change of state to InControl, permitting any actions that you have programmed to occur.

The UPS service that is part of the Windows operating system provides only the first capability, running a command file in response to a power failure. Moreover, it only executes the command file several seconds before the system is shut down. In most instances, this does not provide enough time to be useful, except for the simplest industrial automation purposes. A number of third-party products are available that provide the capability to run both command files.

Using System and User-Defined Variables

Use the RTEngine.PowerFail system variable to coordinate shutdown and startup. The Mode variable automatically created for each program and for each I/O board may also be useful in planning an orderly shutdown. Use the RTEngine.FirstScan and/or the RTEngine.FirstScanOnAutoStart system variables to automate process control restoration by causing program flow to branch to an initialization procedure. Configure the runtime engine to resume execution in the appropriate mode: last mode, paused, stopped, etc., as described in "Checking General Properties of the Runtime Engine."

You can write or force the RTEngine.PowerFail variable from the Watch window, and any programs that use the variable will reflect the new value in their execution. Note that you must reset this variable manually, from the Watch window.

You can define variables (data type = RTECommand) and use the initial values of PowerFailOn and PowerFailOff to simulate power failures for test purposes.

Retentive/Forced Variables and Power Failure

If you check the **Retentive Value** checkbox on a variable's **Symbol Properties** dialog box, you have the option of backing up the value of a variable to the hard disk.

InControl provides three ways by which you can specify for the backup to occur.

- If the runtime engine shuts down during a power failure, the values of retentive and forced variables are copied to the hard disk. When the runtime engine restarts, these values are restored to the variables.

  The values of retentive and forced variables are not saved unless you are using an intelligent UPS with the system and you have configured it to signal InControl of the power failure.

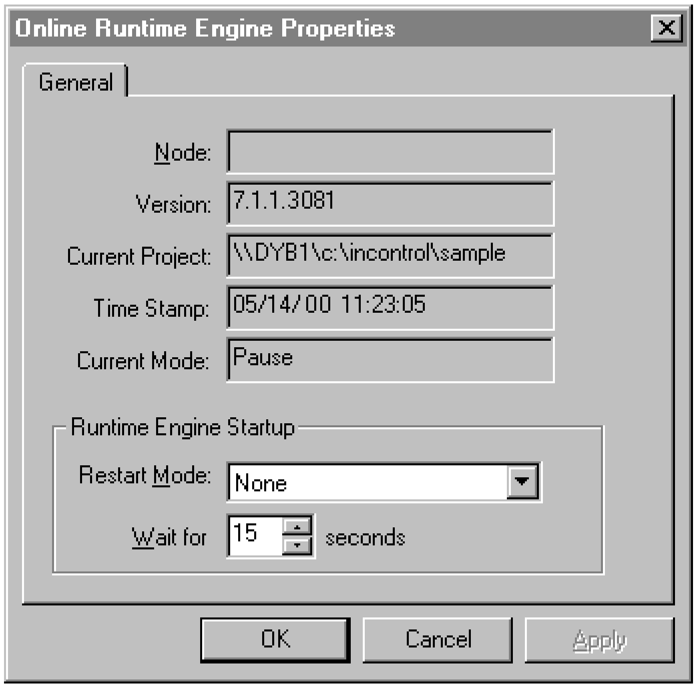
- You can configure InControl to save retentive and forced variables to the hard disk periodically. The default frequency of zero disables this feature. You can change it in the **Runtime Engine Properties** dialog box, described in "Setting Scan Times."

- You can design code in a program to save the value of retentive and forced variables on demand. For a forced variable, both the value and the forced state are saved to the hard disk. Use the following syntax:
  `RTEngine.ExecProjectCommand (SaveRetentive);`

  The values are only restored when the runtime engine is configured to restart automatically (Last, Pause, Run mode) after a system reboot. For more information, see "Restarting Projects Automatically."

Restarting Projects Automatically

In the event that the InControl hardware unit loses power, the runtime engine service starts automatically when power is resumed. The runtime engine returns to the mode specified in the General tab of the **Runtime Engine Properties** dialog box. This dialog box is shown below and described in "Checking General Properties of the Runtime Engine."



Node:
Version: 7.1.1.3081
Current Project: \\DYB1\c:\incontrol\sample
Time Stamp: 05/14/00 11:23:05
Current Mode: Pause

Runtime Engine Startup
Restart Mode: None
Wait for 15 seconds

Restart Mode Field

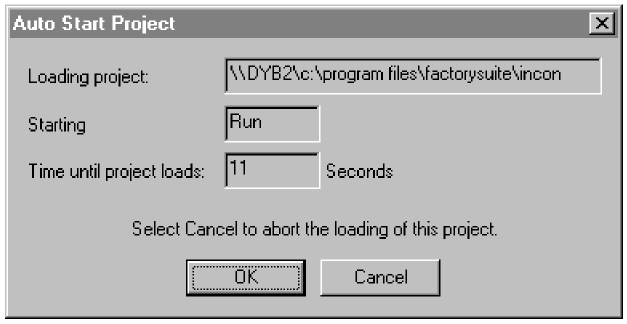
Setting the Restart Mode

Choose from the following restart options, which are selected in the **Restart Mode** field:

- Last—Reload the last project. The runtime engine enters the mode that it was in before the hardware unit was booted.

  Note that when you do a manual shutdown (compared to a shutdown occurring due to power failure) InControl automatically sets the runtime engine to Stop. Therefore, when you reboot, the runtime engine enters the Stop mode. If you want the runtime engine to reload and run the last project, choose the **Run** option.
- None—The runtime engine service starts, but no programs are loaded in the runtime engine.
- Pause—Reload the last project. The runtime engine enters the Pause mode.
- Run—Reload the last project. The runtime engine continues running the project that was running before the hardware unit was booted.

If the runtime engine begins to enter the Run mode after the hardware unit reboots, you have the option of canceling the automatic start of a project. When the runtime engine service restarts, the **Auto Start Project** dialog box appears.



Loading project: \\DYB2\c:\program files\factorysuite\incon
Starting: Run
Time until project loads: 11 Seconds

Select Cancel to abort the loading of this project.

Auto Start Project Dialog Box

If you do not want the project to resume running, click **Cancel** to set the engine to the Stop mode.

Note The value in the **Restart Mode** field determines the mode of the runtime engine after a the system restarts. Individual programs that were in a mode different from the runtime engine before the system restarts will be in the same mode as the runtime engine after a reboot. If you want a program to enter a different mode from that of the runtime engine, use the FirstScanOnAutoStart system variable with appropriate code to set the mode of the program.

Backing Up Retentive/Forced Variables

If the runtime engine shuts down during a power failure, the values of retentive and forced variables are copied to the hard disk.

The values of retentive and forced variables are not saved unless you are using an intelligent UPS with the system and you have configured it to signal InControl of the power failure.

For more information about preparing for power failures, see "Handling Power Failure."

Running Multiple Projects

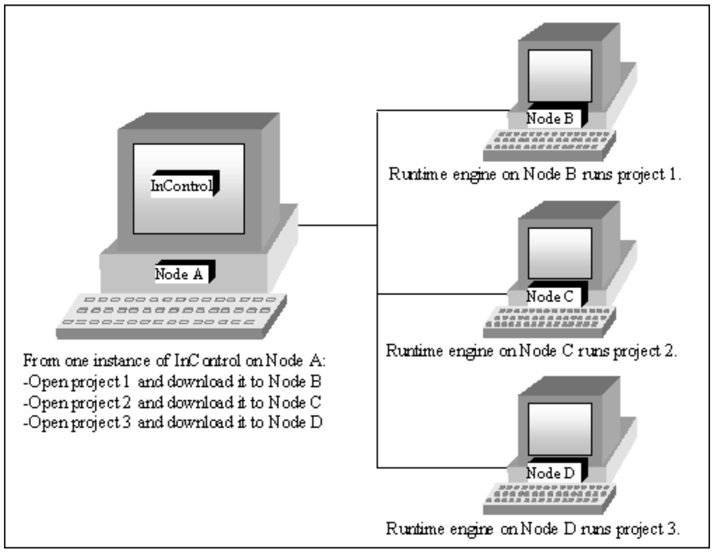
InControl supports distributed control, which enables you to download and run a project on a runtime engine located on another computer. These guidelines apply:

- A project can be opened for editing by one Development environment at a time. This prevents corruption of the project files. One instance of the InControl Development environment can open only one project at a time.
- The Development environment on a node can download a project to the runtime engine located on the connected node.
- The Development environment on any node can connect to a project running on another node, monitor the variables from the Watch window, stop and start the project.
- A node supports one instance of the runtime engine.

Note During installation, you can choose to install only the runtime engine on a node.

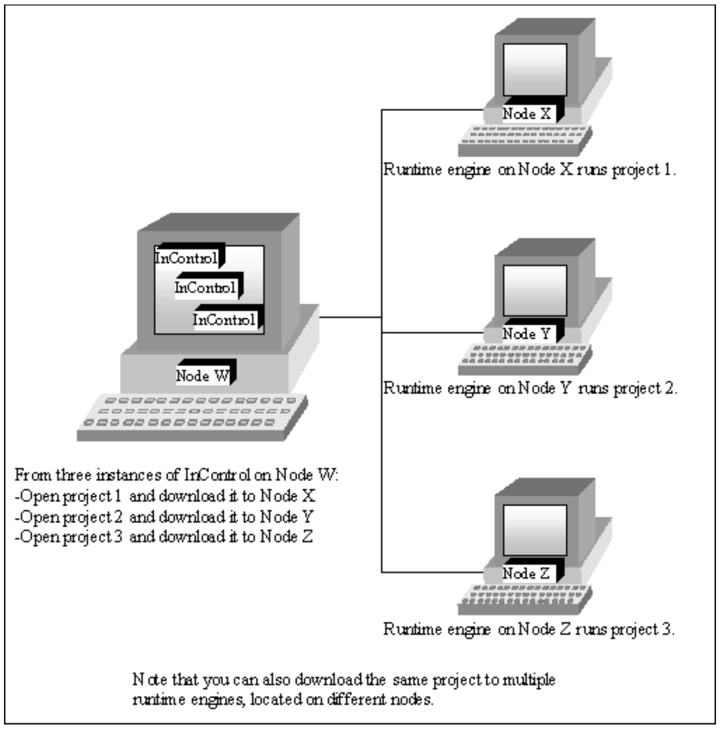
Some examples of distributed control are illustrated in the following figures. Other scenarios are possible.

- On Node A, you can open a project and download it to remote Node B. Then, after switching to a second project (click **Project** on the **File** menu and select the project), you can download this project to remote Node C. You can then open another project and download it to D.


InControl
Node A
From one instance of InControl on Node A:
-Open project 1 and download it to Node B
-Open project 2 and download it to Node C
-Open project 3 and download it to Node D
Node B
Runtime engine on Node B runs project 1.
Node C
Runtime engine on Node C runs project 2.
Node D
Runtime engine on Node D runs project 3.

Multiple Project Example 1

- On Node W, you can run an instance of InControl, open a project, and download it to remote Node X. Then you can run a second instance of InControl. Click **Start** on the **Taskbar**, and point to **Programs**. Under **Wonderware FactorySuite**, select **InControl**. Open a second project and download this project to remote Node Y. Follow the same procedure for node Z.


Node X
Runtime engine on Node X runs project 1.
InControl
InControl
InControl
Node W
Node Y
Runtime engine on Node Y runs project 2.
From three instances of InControl on Node W:
-Open project 1 and download it to Node X
-Open project 2 and download it to Node Y
-Open project 3 and download it to Node Z
Node Z
Runtime engine on Node Z runs project 3.
Note that you can also download the same project to multiple runtime engines, located on different nodes.

Multiple Project Example 2

It is recommended that you plan carefully how you store the source files for a project. If you download a project from node A to a runtime engine on node B, only a binary file is loaded to node B. If you then delete the project on node A, the source code is gone and the project cannot be edited.

Consider having all projects stored or archived on a central server and checking out a copy when you need to make changes. As an alternative, you could make a backup copy of a project on the same node to which you download the project.

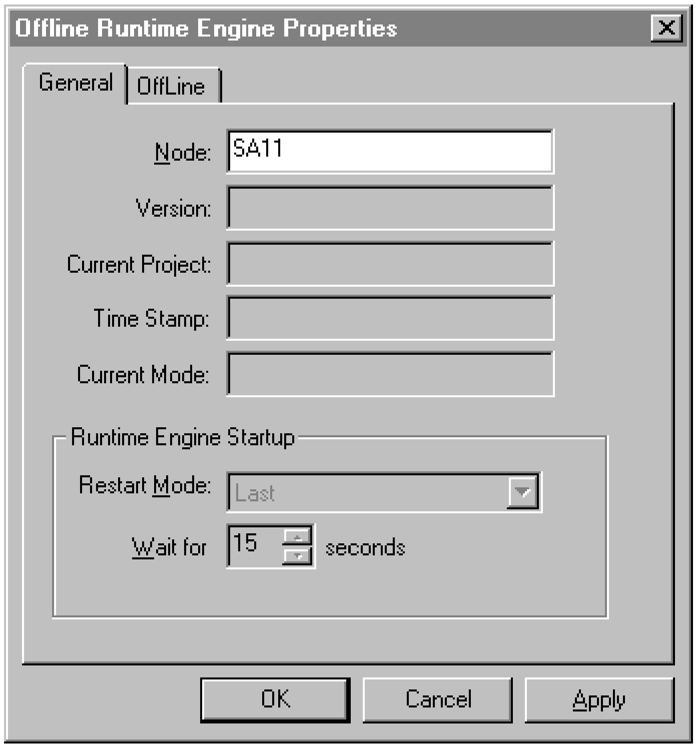
Configuring a Connection to a Remote Node

Designate the remote node on the General tab of the **Runtime Engine Properties** dialog box.

To designate a remote node:

1. Disconnect the development environment from the runtime engine.
2. On the **Runtime** menu, click **Configure**.

3. Select the **General** tab to display the offline properties.



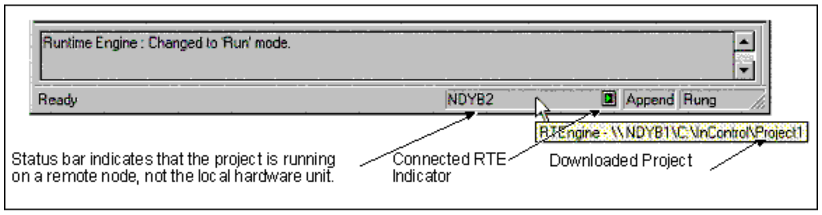
Node: SA11
Version:
Current Project:
Time Stamp:
Current Mode:
Restart Mode: Last
Wait for 15 seconds

General Tab - Node Field

4. Enter the name of the remote node in the **Node** field and click **OK**.

Note Leave the field blank to indicate the local computer.

The next time you click **Connect** on the **Runtime** menu, the development environment connects to the runtime engine located on the remote node.


Runtime Engine : Changed to 'Run' mode.

Ready NDYB2 Append Rung

RTEngine - \\NDYB1\C:\InControl\Project1

Status bar indicates that the project is running on a remote node, not the local hardware unit.
Connected RTE Indicator
Downloaded Project

Remote Node Project

You can enter the machine name or the TCP/IP address of the computer in the Node field of the **Runtime Engine Properties** dialog box. If you use the TCP/IP address and have difficulty connecting to the remote node, enable the Microsoft Network TCP/IP LMHosts lookup.

To enable LMHosts lookup (Windows NT operating system):

1. Click **Start** on the **Taskbar**, then point to **Settings** and click **Control Panel**.
2. Click **Network** and select the Protocols tab.

3. Select **TCP/IP** and click **Properties**.
4. On the WINS Address tab, check the **Enable LMHosts Lookup** checkbox.
5. Reboot the computer for the new setting to become effective.

To enable LMHosts lookup (Windows 2000 operating system):

1. Click **Start** on the **Taskbar**, then point to **Settings** and click **Control Panel**.
2. Right-click your LAN and select **Properties**.
3. Select **Internet Protocol (TCP/IP)** and click **Properties**
4. Click **Advanced**.
5. On the WINS tab, check the **Enable LMHosts Lookup** checkbox.
6. Reboot the computer for the new setting to become effective.

Transferring/Archiving Project Data

When nodes are linked by network connections, downloading a project to a remote node is accomplished by the InControl Development environment. The process is automatic and requires only that you specify a remote node in the **Runtime Engine Properties** dialog box.

The InControl validation utility creates a file called RTEngine.dat that contains all the compiled data of a project. You can use the file for the following purposes:

- Make a copy of RTEngine.dat and store it for archival purposes.
- Copy RTEngine.dat to another node where you can load and run it. This enables you to distribute pre-built projects that do not include the source code. When nodes are not linked, you can transfer RTEngine.dat to a remote node using a manual process.

Note the following considerations.

- It is not necessary to connect to the runtime engine before creating RTEngine.dat.
- When you create RTEngine.dat, it is stored on the local node in the same directory as the project.
- Before you can load the file to the runtime engine (on either the local node or a remote node), you must move it to the working directory of the runtime engine. For Windows NT / 2000 platforms, this is the NT subdirectory, which is immediately below the directory where the RTEngine.exe file is located.

To make a file archive:

1. Validate the project. In the **Validate Project** dialog box, check the **Create Executable Archiveof Project** checkbox. This creates the file archive called RTEngine.dat.
2. Locate the file, which is stored in the same sub-directory as the project.

To load the file in the runtime engine:

1. Verify that the node is the appropriate target platform.
2. Set the runtime engine to the Stop mode.
3. Move the file to the working directory, defined above.
4. Right-click the Runtime Engine Monitor icon and click **Reload Project**.

For example, if the project called BldgCtrl has the following path:

```
C:\Program Files\FactorySuite\InControl\Projects\BldgCtrl
```

you can locate RTEngine.dat in the BldgCtrl sub-directory after you validate the project.

For a Windows NT / 2000 target platform, you must place RTEngine.dat in the following directory before you can load it:

```
..\InControl\NT
```

Using the Watch Window on the Remote Node

You can open and edit a project on a node when the runtime engine on the node is running a different project. If you open the Watch Window on the node, the symbols that you can add to the Watch Window by clicking **Add Symbol** are those defined for the project being edited, not the symbols in the project that is running. However, if you know the names of the symbols used in the project that is running, you can type them into the Watch Window and monitor their status as the program runs.

Note As an alternative to typing in each symbol name within the Watch window, you can create an ASCII file with any text editor and enter the names of the symbols that you want to monitor. See "Monitoring Program Variables" in the "Running a Project" chapter for details about creating the file.

Configuring I/O on the Remote Node

Some of the currently supported I/O drivers allow you to open and edit the I/O configuration of a project that will run on a remote node. The driver must be installed on both the remote node and the local node where you do the configuration. In addition, for some drivers, you may need to install the scanner board in the local node as well as in the remote node.

Note Several I/O drivers provide utilities that you can use to do an automatic configuration and/or run online diagnostics. Some of these drivers require you to use these utilities on the remote node itself. Newer drivers support remote automatic configuration and online diagnostics. To accomplish this, these drivers may download themselves to the remote runtime engine. In this case, the driver enters the Loaded mode (Mode system variable = 8).

Changing System Registry Keys

This section describes changes that you may decide to make in the system registry.

WARNING! Making incorrect changes in the Windows registry can result in unpredictable operation by InControl and I/O devices. This has the potential risk of injury or death to personnel and/or damage to equipment. Always verify that the changes you make in registry keys will not adversely affect the operation of the computer or the applications you intend to run before attempting to control factory field devices. Be aware that Wonderware does not provide technical support for problems that may arise as a result of changes that you make to the registry.

Changing FOE Registry Setting

If the ThreadingModel key for an FOE does not have the correct string value, you cannot install the object in an InControl project. Moreover, the ThreadingModel key must be "Both."for an FOE to operate efficiently in InControl. If the key is "Apartment," access to the properties will be significantly slower. Additionally, it is more likely that the FOE will adversely affect the determinism of the runtime engine. This section describes how to change the ThreadingModel key.

Note Do not change the threading model from "Apartment" to "Both" for ActiveX Controls built using Visual Basic 5.0 or Visual Basic 6.0.

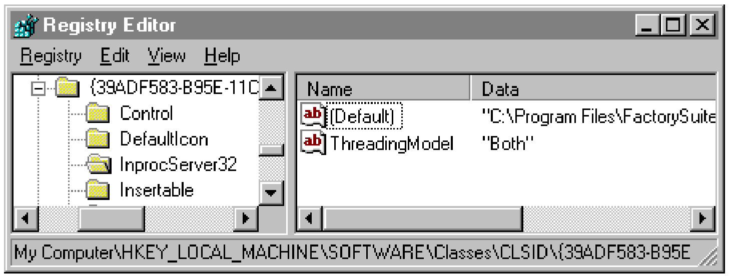
WARNING! If the FOE is not designed to run under the free-threaded model, changing the value of the ThreadingModel may cause the object to run in an unpredictable manner. For example, deadlock may occur, or corrupt values may be read, with the potential risk of death or injury to personnel and/or damage to equipment.

Test and verify that the object runs correctly after changing the ThreadingModel value before using the FOE to control factory floor equipment. Because even exhaustive testing may not reveal problems that can occur in a runtime environment, it is highly recommended that you consult with the developer of the FOE to verify that it will operate correctly with InControl.

To edit the registry:

1. Click **Start** on the **Taskbar**, then click **Run.**
2. Type **regedit** in the **Run** dialog box. The Registry Editor opens.
3. Search for the name of the FOE. For example, to search for the PID FOE, click **Find** on the **Edit** menu and enter "Wonderware PID" in the **Find What** field.
4. Expand the open folder in the registry tree and double-click the **InprocServer32** key.

5. Verify that the string value of ThreadingModel is "Both" as shown below.



Name	Data
(Default)	"C:\Program Files\FactorySuite
ThreadingModel	"Both"

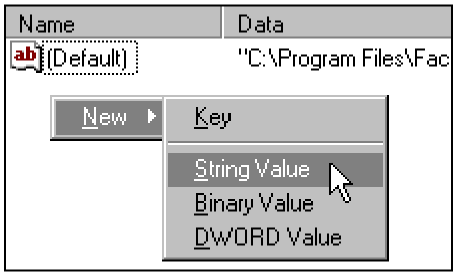
My Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{39ADF583-B95E

Threading Model String Value

6. If you need to edit the value, double-click the value name and enter **Both** in the **Value Data** field.

If a ThreadingModel value does not exist, you can create a new one.

1. Right-click and create a new string value, as shown below.



New Threading Model String Value

2. Enter **ThreadingModel** for the value name.

3. Double-click the new value and enter **Both** in the **Value Data** field.

Displaying Compiler Warnings

If your programs have any code for testing or simulating your process and it writes to I/O input variables, a warning message appears when the programs are validated. You can have the compiler issue an error message instead of a warning. This can be done for a program only, or it can be made specific to a user account and affect all projects.

To make the change at the program level, include the following #pragma statement in an SFC or Structured Text program:

```
#pragma IOWriteError
```

For more information about using the #pragma statement, see "#pragma" in the *InControl STL Reference Manual*.

To make the change for a user account, you need to set a bit in the CPOption key under the following path in the registry.

```
HKEY_CURRENT_USER\
    SOFTWARE\
        WONDERWARE\
            INCONTROL
```

WARNING! Making incorrect changes in the Windows registry can result in unpredictable operation by InControl and I/O devices. This has the potential risk of injury or death to personnel and/or damage to equipment. Always verify that the changes you make in registry keys will not adversely affect the operation of the computer or the applications you intend to run before attempting to control factory field devices. Be aware that Wonderware does not provide technical support for problems that may arise as a result of changes that you make to the registry.

If the CPOption key does not exist, create a new (DWORD value) key and label it CPOption.

For the compiler to issue a warning, set bit 6 to 0. This is the default value. For the compiler to issue an error, set bit 6 to 1.

For example, assuming no other bits are set to 1, write 20H to the CPOption key to have the compiler issue error messages instead of warnings. You can have no messages issued at all, but this can only be done with the #pragma statement.

Issuing Runtime Engine Commands

Two functions are associated with the runtime engine: ExecProgramCommand and ExecProjectCommand. To execute these functions, call them from another InControl program (Structured Text or SFC). The command values used by these functions are summarized in the table below.

Project Information Command Values

Function	**Command Value**	**Command Result**
ExecProgramCommand	Step	Single step a program.
ExecProgramCommand or ExecProjectCommand	ClearFaults	Clear faults (program or project).
ExecProgramCommand or ExecProjectCommand	ReportStatus	Provides runtime engine status data, such as current project, time stamp, scan time, mode, processor utilization, faulted programs, I/O faults, etc.
ExecProjectCommand	PowerFailOn	Simulates a power failure by setting the RTEngine.PowerFail symbol to TRUE.
ExecProjectCommand	PowerFailOff	Simulates a recovery from a power failure by setting the RTEngine.PowerFail symbol to FALSE.
ExecProjectCommand	SaveRetentive	Save retentive values for a project.

For a program, use the following syntax:

```
RTEngine.ExecProgramCommand("programname",<commandvalue>)
```

For a project, use the following syntax:

```
RTEngine.ExecProjectCommand (<command value>)
```

Value/Time/Quality Support

The runtime engine reports Value/Time/Quality (VTQ) information. Applications, such as InTouch and IndustrialSQL, that can connect to InControl with the SuiteLink or DDE 3 protocol, are able to obtain this data.

Runtime engine data values, such as memory tags, have a timestamp based on the last scan. The quality is reported as follows:

- Good indicates that the runtime engine is not in the Fault mode.
- Bad indicates that the runtime engine is in the Fault mode.

The timestamp and quality for data values from an instruction processor (IP), such as an ActiveX control or an I/O driver, is determined by the IP. Quality is determined as follows, based on the program mode:

- Stopped mode—Quality is reported as Bad.
- Fault mode—Quality is reported as Bad and the device has failed.
- Idle—Quality is reported as Uncertain, and Stale.

When you are using the DDE 3 protocol, updates are sent to clients when either the value or the quality changes. However, if you are using the DDE 2 protocol, only value changes are transmitted.

Entering Event Viewer Settings

The Windows Event Viewer monitors events as they occur in your hardware unit. It receives messages from the runtime engine and I/O drivers, for example. If the buffer space that you have allocated for holding the messages becomes full, preventing the Event Viewer from receiving messages, the runtime engine service may stop.

WARNING! Allowing the Event Viewer log to become full and unable to receive system messages may cause the runtime engine service and all programs to stop unexpectedly. This can result in unpredictable operation by field devices and cause death or injury to personnel and/or damage to equipment. If you receive a message from the Event Viewer indicating that it is becoming full, take action to clear the log. It is recommended that you choose the Overwrite Event as Needed option in the Event Log Settings dialog box.

A P P E N D I X A

Reserved Words

The words listed in the following table are InControl reserved words. Avoid using any of these reserved words in your program code.

InControl Reserved Words

ABORT_ALL	ABS	ABTAL
ACOS	ACTION	ADD
AND	ANDN	ANDT
ANDTN	ANY	ANY_BIT
ANY_DATE	ANY_INT	ANY_NUM
ANY_REAL	APPENDFILE	ARRAY
ARRAY_TO_STRING	ASIN	AT
ATAN		
BCD_TO_INT	BEGIN	BEGIN_IL
BOOL	BREAK	BY
BYTE		
CAL	CALC	CALCN
CASE	CLOSEFILE	CONCAT
CONFIGURATION	CONSTANT	COPYFILE
COS	CTD	CTU
CTUD		
D	DATE	DATE_AND_TIME
DATE_TO_REAL	DATE_TO_STRING	DELETE
DELETEFILE	DINT	DIV
DO	DS	DT
DWORD		
ELSE	ELSEIF	ELSIF
END	END_ACTION	END_CASE
END_ CONFIGURATION	END_FOR	END_FOR_NOWAIT
END_FUNCTION	END_FUNCTION_ BLOCK	END_IF
END_IL	END_PROGRAM	END_REPEAT
END_REPEAT_ NOWAIT	END_RESOURCE	END_STEP
END_STRUCT	END_TRANSITION	END_TYPE
END_VAR	END_WHILE	END_WHILE_ NOWAIT
EQ	EXIT	EXP
EXPT		

F_EDGE	F_TRIG	FALSE
FB	FILE	FIND
FOR	FROM	FUNCTION
FUNCTION_BLOCK		
GE	GLOBAL	GOTO
GT		
IF	INCLUDE	INITIAL_STEP
INSERT	INT	INT_TO_BCD
INT_TO_REAL	INT_TO_STRING	INTERVAL
JMP	JMPC	JMPCN
L	LD	LDN
LE	LEFT	LEN
LIMIT	LINT	LN
LOG	LREAL	LT
LWORD		
MACROSTEP	MAX	METHOD
MID	MIN	MOD
MODE	MOVE	MSGWND
MUL	MUX	
N	NE	NEWFILE
NOT	NOW	
OF	ON	OPENFILE
OR	ORN	ORT
ORTN		
P	POW	PRIORITY
PROGRAM	PW	
R	R_EDGE	R_TRIG
READ_ONLY	READ_WRITE	READFILE
REAL	REAL_TO_DATE	REAL_TO_INT
REAL_TO_STRING	REAL_TO_TIME	REPEAT
REPLACE	RESOURCE	RET

RETAIN	RETC	RETCN
RETURN	REWINDFILE	RIGHT
ROL	ROR	RS
RUNG		
S	SCAN	SD
SEL	SHL	SHR
SIN	SINGLE	SINT
SL	SQRT	SR
ST	START	STEP
STN	STRING	STRING_TO_ARRAY
STRING_TO_DATE	STRING_TO_INT	STRING_TO_REAL
STRING_TO_TIME	STRUCT	STT
STTN	SUB	
T	TAN	TASK
THEN	TIME	TIME_OF_DAY
TIME_TO_REAL	TIME_TO_STRING	TMR
TO	TOD	TODAY
TOF	TON	TP
TRANS	TRANSITION	TRUE
TRUNC	TYPE	
UDINT	UINT	ULINT
UNTIL	USINT	
VAR	VAR_ACCESS	VAR_EXTERNAL
VAR_GLOBAL	VAR_IN_OUT	VAR_INPUT
VAR_OUTPUT		
WHILE	WITH	WORD
WRITEFILE		
XOR	XORN	

A P P E N D I X B

Data Types

This appendix provides a quick reference to the data types supported by InControl. For more information about the data types, see Chapter 6, "Defining Variables."

Contents

Data Type Categories

Data Types and Categories

All Types	Group	SubGroup	Data Type
ANY	ANY_NUM	ANY_REAL	LREAL
			REAL
		ANY_INT	DINT
			INT
			SINT
			DWORD [1]
			WORD [1]
			BYTE
	ANY_BIT		DWORD
			WORD
			BYTE
			BOOL
	ANY_DATE		DT (date and time)
			DATE
			TOD
			TIME
			TMR [2]
			FILE [2]
			STRING
			User-Defined
			ANY
Enumeration			RTEMODE

Note The LINT, ULINT, and LWORD data types are not currently supported by InControl.

1 The UDINT, UINT, and USINT data types are equivalent to the DWORD, WORD and BYTE data types respectively. An InControl enhancement to the ANY_BIT data types makes the UDINT, UINT, and USINT data types unnecessary.

2 Enhancement to the IEC 61131-3 specification.

WARNING! IEC-61131 does not support the combination of signed and unsigned numbers (ANY_NUM data types) in a math calculation. If you do combine signed and unsigned numbers, the results of the math operation may not be what you expect, which may have the potential risk of death or injury to personnel and/or damage to equipment. Avoid using expressions that combine signed and unsigned numbers.

Data Type Ranges

Data Type	Range
LREAL	-1.79769313486231 E308 (negative) to +1.79769313486231 E308 (positive) and includes zero.When you communicate with the runtime engine using a SuiteLink/DDE interface, includingInTouch, 64 bit LREAL data types are transmitted at 32-bit precision.
REAL	-3.402823 E38 (negative), to +3.402823 E38 (positive), and includes zero.
DINT	-2147483648 to +2147483647
INT	-32768 to +32767
SINT	-128 TO +127
DWORD	0 to 4294967295
WORD	0 to 65535
BYTE	0 to 255
BOOL	TRUE or FALSE
DT	Format: DATE_AND_TIME | date_and_time | DT | dt#YYYY-MM-DD-HH:MM:S.S YYYY (100-2100) = year MM (1-12) = month DD (1-31) = day of the month HH (0-23) = hour MM (0-59) = minute S.S (0.0-59.0) = seconds (real number)
DATE	Format: DATE | date | D | d#YYYY-MM-DD YYYY (100-2100) = year MM (1-12) = month DD (1-31) = day of the month

Data Type	Range	
FILE	Has three system input variables, which are identified by the function control block name (fcb) plus an extension.	
	fcb.ACCESS	Byte variable specifies read/write status of file after it opens. 0 = (default) file is open for read/write operations. 1 = file is open for read-only operations. 2 = file is open for write-only operations.
	fcb.APPEND	Boolean variable specifies whether data can be appended to the file after it opens.Only valid when file is open with write status. That is, the ACCESS variable = 0 or 2. TRUE = data will be appended to the file. FALSE = (default) data cannot be appended to the file.
	fcb.SHARE	Byte variable specifies how other applications can access the file after it is open. 0 = (default) other applications access file for read-write operations. 1 = other applications access file for read-only operations. 2 = other applications access file for write-only operations. 3 = other applications cannot access the file.
	Has eight system output variables, which are identified by the function control block name (fcb) plus an extension.	
	fcb.OPEN	Boolean variable that indicates the file has been opened.
	fcb.BUSY	Boolean variable that indicates the file is being accessed.
	fcb.EFLAG	Boolean variable indicates when an error occurs.
	fcb.RDN	Boolean variable that indicates a read operation has been completed.
	fcb.WDN	Boolean variable that indicates a write operation has been completed.
	fcb.CLSD	Boolean variable that indicates a file has been closed.
	fcb.EOF	Boolean variable that indicates the system encountered an End Of File.
	fcb.ERR	Integer variable that contains the error code if an error occurs.

Data Type	Range
STRING	Format: ‘ Character_representation ’ Character_representation = any printable character up to 1024 characters. A $ followed by two hexadecimal digits, enclosed in single quotation marks, is interpreted as the hexadecimal representation of the eight-bit character code. Example: ‘ $41 $42 $43 ’ is interpreted as A B C. To designate these characters, use the following formats: Dollar sign = ‘ $$ ’ Single quote = ‘ $’ Line feed = ‘ $L ’ or ‘ $l ’ New line = ‘ $N ’ or ‘ $n ’ Form feed = ‘ $P ’ or ‘ $p ’ Carriage return = ‘ $R ’ or ‘ $r ’ Tab = ‘ $T ’ or ‘ $t ’

TIME	Format: TIME | time | T | t #Days | Hours | Minutes | Seconds | Milliseconds
	Days = xD, where x = an integer (0-10675199), e.g. 5D.
	Hours = xH,
	where x = an integer (0-23), e.g., 7H.
	Minutes = xM, where x = an integer (0-59), e.g., 5M.
	Seconds = xS, where x = an integer (0-59), e.g., 5S.
	Milliseconds = x MS, where x = an integer (0-999), e.g., 200MS.
	Example: T#1D20H13M15S500MS is interpreted as 1 day, 20 hours, 13 minutes, 15 seconds, and 500 milliseconds.
	Example: t#1d2h31M6s5ms is interpreted as 1 day, 2 hours, 31 minutes, 6 seconds, and5 milliseconds.Format: TIME | time | T | t #Days | Hours | Minutes | Seconds | Milliseconds
	The maximum time that you can enter through a program, HMI, or Watch window is +/- 10675199d2h48m5s477ms. Values greater than this that are entered through the Watch window or the Symbol Manger will generate an error.

TMR	Has four system variables, which are identified by the timer name plus an extension. Tmr_name.PT contains the preset time value. This variable is retentive. That is, it retains its value during a power loss. Tmr_name.EN starts/stops the TMR and is a BOOLEAN data type. Tmr_name.ET contains the elapsed time of the TMR in seconds. Tmr_name.Q contains the TMR output.
TOD	Format: TIME_OF_DAY | time_of_day | TOD | tod#HH:MM:SS HH (0-23) = hour MM (0-59) = minute S.S (0.0-59.0) = second
User-Defined	Consists of a set of data types that function as a group. The members do not all have to be the same data type. Ranges are determined by the data type for each individual member.

Data Type	Range
ANY	Range depends on the data type that the symbol currently has.
RTEMODE	Valid values are: COMPLETE, FAULT, LOADED, PAUSE, PROGRAM, RUN, SCAN, STOP, UNKNOWN

A P P E N D I X C

Monitoring Data By DDE/SuiteLink

This appendix provides information about how to monitor InControl variables from other applications.

Contents

Overview

You can monitor InControl variables from another application, such as InTouch or Excel, for example. You can use either the DDE or the SuiteLink protocol. This appendix describes how to use DDE and InControl operates as a DDE server.

If the application is on a remote node, you need to configure DDE Shares for that node.

For information about using SuiteLink, which allows InControl to operate as a SuiteLink client, see the *Wonderware InControl SuiteLink Version 2 User's Guide*.

Note Some applications, such as Excel, may not be able to monitor the elements in arrays or the members of structures.

Monitoring Variables from InTouch

InTouch can receive data from InControl when you create DDE items in the InTouch Tagname Data Dictionary.

For detailed instructions about how to use a DDE connection to monitor anInControl variable from InTouch, see the *InTouch User's Guide*.

When you create a DDE item, you need the following information:

For the application name, use RTEngine.

For the topic, use TagName.

For the item name, use the name of the variable that you want to monitor, following these rules:

- To reference a global InControl variable, use only the name of the variable.
- To reference a local InControl variable, precede the variable name with the program name followed by a period. For example, if the program name is SeamWeld, and the variable name is weld_done, use Seamweld.weld_done as the DDE item name.
- To reference a user-defined data type, precede the name of the member with the name of the variable of the user-defined data type followed by a period. For example if the user-defined data type variable is Device_Status and the member of the Device_Status variable that you want to monitor is Speed, use Device_Status.Speed as the DDE item name.

When you display SuiteLink tags in the InControl Watch window, you may observe that their values are changing, even when the tags have been forced. This is due to the reflection of pokes by a SuiteLink server to other clients. You can prevent this by making a change to the Windows registry.

WARNING! Making incorrect changes in the Windows registry can result in unpredictable operation by InControl and I/O devices. This has the potential risk of injury or death to personnel and/or damage to equipment. Always verify that the changes you make in registry keys will not adversely affect the operation of the computer or the applications you intend to run before attempting to control factory field devices.
Be aware that Wonderware does not provide technical support for problems that may arise as a result of changes that you make to the registry.

To change the registry setting:

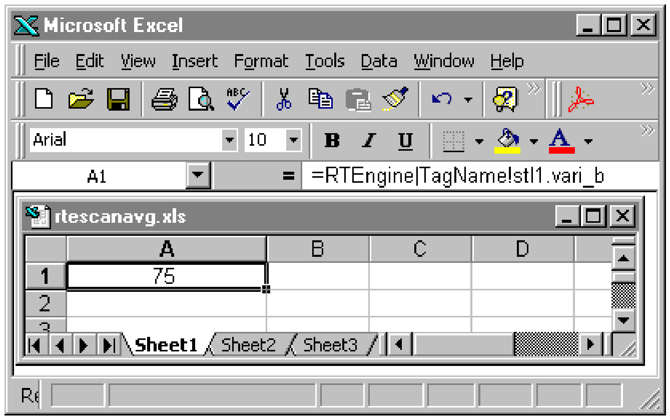
1. Edit the registry at the following location.
```
HKEY_LOCAL_MACHINE\
    SOFTWARE\
        WONDERWARE\
            INCONTROL
```
2. Create a new key called SuiteLinkReflectPokes. It must be a DWORD.
3. Assign the key a value of 0. This prevents the server from reflecting pokes received via SuiteLink to other clients.

Monitoring Variables from Excel

Excel can read data from InControl when you create a DDE remote reference in a cell.

Note Some applications, such as Excel, may not be able to monitor the elements in arrays or the members of structures.

In the following figure, the value of the InControl local variable Vari_b (program STL1) is monitored in cell A1.



=RTEngine|TagName!stl1.vari_b

	A	B	C	D
1	75			
2				

Excel Example Click to see figure.

The formula in A1 is

`=RTEngine|TagName!STL1.Vari_b`

where

= is the Excel operator Equal To

`RTEngine` is the application name

| is the pipe symbol used to separate the application name from the topic name

`TagName` is the topic name (spelled as shown in the example)

! is the delimiter between the topic name and the variable name, and

`Vari_b` is the name of the InControl variable being monitored in program `STL1`.

You can list any variable name here.

To reference a local InControl variable, precede the variable name with the program name followed by a period. For example, if the program name is PowerUp, and the variable name is lights_on, use PowerUp.lights_on as the DDE item name.

To reference a user-defined data type, precede the name of the member with the name of the data type followed by a period. For example if the user-defined data type is Time and the member of the Time structure that you want to monitor is Hour, use Time**.**Hour as the DDE item name.

This formula is dynamic and updates the value in A1 continually. For more information about entering formulas in an Excel spreadsheet, see the Excel user documentation.

You must configure DDE Shares for the application "RTEngine."

APPENDIX D

Extensions to IEC 61131-3

This appendix describes enhancements and other extensions to the IEC 61131-3 specification. InControl complies with IEC 61131-3 except where noted in this appendix and in the InControl Language Editors manual.

Contents

Data Types

For more information about data types, see the "Defining Variables" chapter.

Unsupported Data Types

InControl does not support the following data types, which are defined in the IEC 61131-3 specification.

- LINT
- ULINT
- LWORD

The UDINT, UINT, and USINT data types are equivalent to the DWORD, WORD, and BYTE data types, respectively. An InControl enhancement to the ANY_BIT data types makes the UDINT, UINT, and USINT data types unnecessary.

Data Type Conversion

InControl does a limited automatic conversion of data types. It is not necessary to use the IEC-specified INT_TO_REAL and REAL_TO_INT functions, because conversion is handled internally by the RLL MOVE function block and the Structured Text Assignment statement.

For REAL to integer data conversions, InControl truncates the result. If you need to round off the result, add 0.5 to the REAL value before converting it to an integer. You can also create a user-defined function to do rounding operations as needed.

For more information about how InControl handles data type conversion, see "Data Type Conversion" in the "Defining Variables" chapter.

Parameters Specific to InControl

InControl parameters, which are implementation specific, are listed in the following table.

Implementation-Dependent Parameters

Clause	Parameter	Implementation
1.5.1	Error handling procedures.	See "Error Conditions."
2.1.1	National characters used.	None
2.1.2	Maximum length of identifiers.	100
2.1.5	Maximum comment length.	Not limited
2.2.3.1	Range of values of duration.	Range of REAL in ms
2.3.1	Range of values for variables of TIME data type.	Range of REAL in ms
2.3.1	Precision of representation of seconds in data types TIME_OF_DAY and DATE_AND_TIME.	Range of REAL in ms
2.3.3	Maximum number of array subscripts.	1
2.3.3	Maximum array size.	Limited by space used by all symbols.
2.3.3	Maximum number of structure elements.	Limited by space used by all symbols.
2.3.3	Maximum structure size.	Limited by space used by all symbols.
2.3.3	Maximum number of variables per declaration.	Limited by space used by all symbols.
2.3.3.1	Maximum number of enumerated values.	n/a
2.3.3.2	Default maximum length of STRING variables.	1024
2.3.3.2	Maximum allowed length of STRING variables.	1024
2.4.1.1	Maximum number of hierarchical levels.	n/a
2.4.1.1	Logical or physical mapping.	Logical
2.4.1.2	Maximum number of subscripts.	1
2.4.1.2	Maximum range of subscript values.	Not limited.
2.4.1.2	Maximum number of levels of structures.	Not limited.
2.4.2	Initialization of system inputs.	Same as all variables.
2.4.3	Maximum number of variables per declaration.	n/a

Clause	Parameter	Implementation
2.5.1.1	Method of function representation.	Names and function blocks.
2.5.1.3	Maximum number of function specifications.	Limited by space used by all symbols.
2.5.1.5	Maximum number of inputs of extensible functions.	2
2.5.1.5.1	Effects of type conversions on accuracy.	n/a
2.5.1.5.2	Accuracy of functions of one variable.	Same as C library function accuracy.
2.5.1.5.2	Implementation of arithmatic functions.	Same as C library function accuracy.
2.5.2	Maximum number of function block specifications and instantiations.	n/a
2.5.2.3.3	PVmin, PVmax of counters.	Of DINT type.
2.5.3	Program size limitations.	Not limited.
2.6.2	Precision of Step elapsed time.	ms
2.6.2	Maximum number of Steps per SFC.	Not limited.
2.6.3	Maximum number of Transitions per SFC and per Step.	Not limited.
2.6.4	Action control mechanism.	Graphical declaration in RLL language.
2.6.4.2	Maximum number of Action blocks per Step.	Not limited.
2.6.5	Graphic indication of Step state.	Step is highlighted.
2.6.5	Transition clearing time.	One scan.
2.6.5	Maximum width of diverge/converge constructions.	Not limited.
2.7.1	Contents of resource libraries.	n/a
2.7.2	Maximum number of tasks.	Not limited.
2.7.2	Task interval resolution.	ms
2.7.2	Preemptive or non-preemptive scheduling.	Non-preemptive.
3.3.1	Maximum length of expressions.	Not limited.
3.3.1	Partial evaluation of Boolean expressions.	Full evaluation.
3.3.2	Maximum length of statements.	Not limited.
3.3.2.3	Maximum number of CASE selections.	Not limited.
3.3.2.4	Value of control variable upon termination of FOR loop.	One beyond final value.
4.1.1	Graphic/semigraphic representation.	Graphic.

Clause	Parameter	Implementation
4.1.1	Restrictions on network topology.	Match convergences and divergences.
4.1.3	Evaluation order of feedback loops.	n/a

Error Conditions

The error conditions defined in IEC 61131-3 are listed in the following table. The method of detection (program preparation or program execution) is listed.

Clause	Error Condition	Detection
2.3.3.1	Value of a variable exceeds the specified subrange.	Preparation and Execution.
2.4.2	Length of initialization list does not match number of array entries.	n/a
2.5.1.5.1	Type conversion errors.	n/a
2.5.1.5.2	Numerical result exceeds range for data type.	Execution.
2.5.1.5.2	Division by zero.	Execution.
2.5.1.5.4	Mixed input data types to a selection function.	n/a
2.5.1.5.4	Selector (K) out of range for MUX function.	n/a
2.5.1.5.5	Invalid character position specified.	Execution.
2.5.1.5.5	Result exceeds maximum string length.	Execution.
2.5.1.5.6	Result exceeds range for data type.	Execution.
2.6.2	Zero or more than one initial Steps in SFC network.	Preparation.
2.6.2	User program attempts to modify Step state or time.	Preparation.
2.6.2.5	Simultaneously true, non-prioritized Transitions in a selection divergence.	n/a
2.6.3	Side effects in evaluation of Transition condition.	n/a
2.6.4.5	Action control contention error.	Execution.
2.6.5	Unsafe or unreachable SFC.	Preparation.
2.7.1	Data type conflict in VAR_ACCESS.	Preparation.
2.7.2	Tasks require too many processor resources.	Execution.
2.7.2	Execution deadline not met.	Execution.
2.7.2	Other task scheduling conflicts.	Execution.
3.2.2	Numerical result exceeds range for data type.	Execution.
3.3.1	Division by zero.	Execution.
3.3.1	Invalid data type for operation.	Execution.
3.3.2.1	Return from function without value assigned.	n/a
3.3.2.4	Iteration fails to terminate.	None.
4.1.1	Same identifier used as connector label and element name.	Preparation.
4.1.4	Uninitialized feedback variable.	n/a

Note InControl does not always indicate overflow and range errors to the user at execution time. This exception to the IEC 61131 specification was made to improve performance. The user must customize error checking for critical code segments to suit the control application.

A P P E N D I X E

Keyboard Shortcuts

This appendix lists the shortcut key combinations for the InControl tools and menu options. These shortcut keys are based on the U.S. keyboard. If you are using a keyboard for a different language, you may need to change your shortcut key combinations.

Contents

General Operations

Menu Bar Shortcuts

Operation	Key or Key Combination
Select All.	Ctrl A
Open new program.	Ctrl N
Open existing program.	Ctrl O
Print selected program.	Ctrl P
Save selected program.	Ctrl S
Copy selected item to clipboard.	Ctrl C
Paste contents of clipboard.	Ctrl V
Cut selected item and place it in the clipboard.	Ctrl X
Undo last operation.	Ctrl Z
Redo or repeat last operation.	Ctrl Y
Find selected item.	Ctrl F
Find next occurrence of selected item.	F3
Replace selected item.	Ctrl H
Delete selected item.	DEL
Open Symbol Manager.	Ctrl T
Display Online Help.	F1
Set runtime engine to Stop mode.	Ctrl Break
Step program.	F10
Validate project.	F4
Validate program.	Shift F4
Run project.	F5
Run program.	Shift F5
Download project.	Ctrl F5
Download program.	Shift Ctrl F5
Pause project.	F7
Pause program.	Shift F7
Single scan project.	F8
Single scan program.	Shift F8
Toggle breakpoints.	F9
Clear all breakpoints.	Shift Ctrl F9

Window Operations

Operation	Key or Key Combination
Switch focus between Project window, Watch window, Output window, and Development/Runtime window.	ALT F6
Display Project Window.	Ctrl J
Display Block Palette.	Ctrl B
Display Watch Window.	Ctrl W
Switch focus between programs open in the editor window.	Ctrl F6

Project Window

File and Window Shortcuts

Operation	Key or Key Combination
Delete selected program.	DEL
Open selected program for editing.	Enter
Display properties for selected program.	ALT Enter
Display properties for selected program.	F2 *or* Alt Enter

Output Window

Window Shortcuts

Operation	Key or Key Combination
Copy selected text to the clipboard.	Ctrl C
Clears window of all information, whether or not any text is selected.	DEL

Project Manager

File Shortcuts

Operation	Key or Key Combination
Create new project.	Ctrl N
Open selected project.	Enter *or* Ctrl O
Delete selected project.	DEL
Display properties for selected project.	F2 *or* Alt Enter

Watch Window

Toolbar Shortcuts

Operation	Key or Key Combination
Copy selected item to clipboard.	Ctrl C
Paste contents of clipboard.	Ctrl V
Cut selected item and place it in the clipboard.	Ctrl X
Remove selected symbol.	DEL
Modify selected value.	Ctrl M
Unforce selected symbol.	Ctrl U
Unforce all symbols.	Ctrl A
Set number base for selected symbol to binary.	Ctrl B [1]
Set number base for selected symbol to octal.	Ctrl O [1]
Set number base for selected symbol to decimal.	Ctrl D [1]
Set number base for selected symbol to hexadecimal.	Ctrl H [1]
Insert row.	INS
Swap focus between symbol list and Watch Window table list.	TAB
Display tables in Watch Window table list.	ALT Down Arrow
Expand a collapsed tree of symbols.	Right Arrow
Move cursor to the right from a single symbol or an expanded tree of symbols.	Right Arrow
Collapse an expanded tree of symbols.	Left Arrow
Double-click any cell.	F2
1 If an array or structure is expanded and the parent element has focus, the change in number base is applied to all elements of the array or structure.	

Program Editors

RLL Shortcuts

Operation	Key or Key Combination
Insert a branch.	Shift B
Insert a new rung.	Shift R
Insert a contact.	Shift C
Insert a coil.	Shift O
Insert a function or function Block	Shift F
Edit selected rung element.	F2
Go to specified rung.	Ctrl G
Display Block Palette.	Ctrl B

Structured Text Shortcuts

Operation	Key or Key Combination
Go to specified line or bookmark.	Ctrl G
Select all text.	Ctrl A
Mark current line.	Ctrl L
Open the Symbol Manager.	Ctrl T
Open the Symbol Manager. Select a symbol and click **OK** to insert the selected symbol into the program.	Shift Ctrl T
Insert a function or function Block	Shift F
Display Block Palette.	Ctrl B

FOE Shortcuts

Operation	Key or Key Combination
Close configuration without saving changes entered in the current tab.	ESC
Edit FOE properties.	ALT Enter

SFC Shortcuts

Operation	Key or Key Combination
Insert an Action.	Shift A
Insert a Comment.	Shift C
Insert a Select Divergence.	Shift D
Insert a Jump.	Shift J
Insert a Label.	Shift L

Insert a Macro Step.	Shift M
Insert a Loop.	Shift O
Insert a Simultaneous Divergence.	Shift P
Insert a Step.	Shift S
Edit Step properties.	ALT Enter
Insert a Transition.	Shift T
Insert a Library Step.	Shift Y
Edit selected program element.	F2
Open Action Manager.	Ctrl M
Close an Action.	ESC
Open Transition Manager	Ctrl R
Display Block Palette.	Ctrl B

Symbol Manager

Object and Window Shortcuts

Operation	**Key or Key Combination**
Open the **Export Select Symbols** dialog box.	ALT E
Open the **Open** dialog box to import symbols.	ALT I
Open the **Print Select Symbols** dialog box.	ALT P
Select the next higher level of scoping.	Backspace
Add a symbol.	Insert
Delete a symbol.	DEL
Display properties for selected symbol.	ALT Enter

Symbol Picker

Keyboard Shortcuts

Operation	**Key or Key Combination**
Display defined symbols.	Ctrl Space
Create symbol (only when drop-down is shown)	ALT C
Browse for symbol (only when drop-down is shown)	ALT B

Index

Wonderware®

InControl™ Language Editors User's Guide

Revision H

Last Revision: July 2004

Invensys Systems, Inc.

Invensys Systems, Inc.
26561 Rancho Parkway South
Lake Forest, CA 92630 U.S.A.
(949) 727-3200
http://www.wonderware.com

Contents

C H A P T E R 1

Relay Ladder Logic Program Elements

This chapter introduces the Relay Ladder Logic (RLL) editor and how to use it to create a new RLL program.

Contents

Power Flow - Solving Simple Contact/Coil Logic

The RLL program is composed of two vertical lines (power rails) that are connected by one or more horizontal lines (the RLL rungs). The left rail represents the power source, and the right rail represents the sink. The basic RLL program elements, contacts and coils, are located on the rungs and represent the actual hardware components (limit switches, solenoid coils, lights, etc.) and also single-bit internal memory locations.

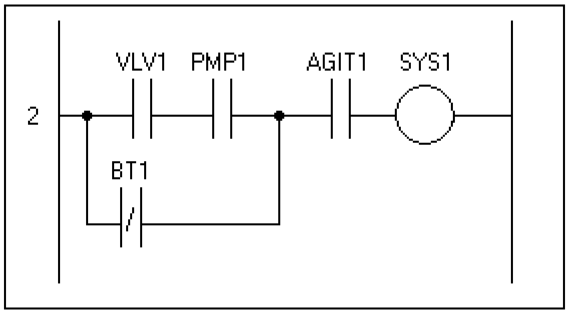
After the system writes to the physical outputs, it reads physical inputs and then solves the RLL logic. Power flow, and solving of the logic of the program, in an RLL program is always from top to bottom, and from left to right.

For more information about the InControl timeline, see "Runtime Engine Timeline" in the "InControl System Administration" chapter.

In Example 1, shown in the following figure, power flow begins at the left power rail, and if contact VLV1 is on, power flow continues to contact PMP1. The three contacts VLV1, PMP1, and AGIT1 are all in series and represent the logical ANDing of the three contacts. Contact BT1 is in parallel with contacts VLV1 and PMP1, representing the logical OR of BT1 with VLV1 and PMP1. If contact BT1 is on, power flow continues to AGIT1, even if VLV1 is not on. If power flow continues to coil SYS1, then SYS1 turns on the and circuit is complete to the right power rail.

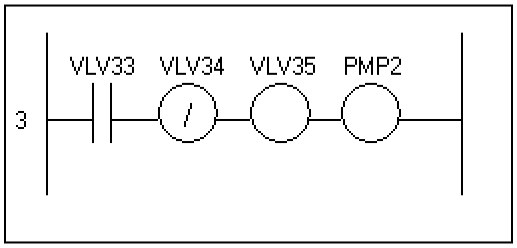
The logic for Example 1 is the following:

```
SYS1 := ((VLV1 AND PMP1) OR NOT BT1) AND AGIT1;
```



Power Flow Example 1

In Example 2, note that coil PMP2 always turns on when contact VLV33 is on, regardless of the status of coils VLV34 and VLV35.

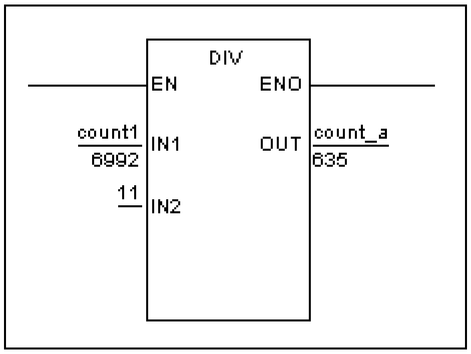


Power Flow Example 2

Power Flow - Function Blocks

RLL function blocks are preprogrammed packages that can also be placed on an RLL rung. They provide a mechanism for solving more complex problems not easily handled by contacts and coils: math operations, logic functions, timers, etc. Function block inputs receive power flow from the rung and transfer power flow to the next element on the rung through their outputs. They can also read and write data through internal inputs and outputs.

In the figure below, the division function block (DIV) is enabled by its rung input EN. It receives divisor and dividend data through two internal inputs (IN1, IN2). The function block writes the quotient to an internal output (OUT) and transfers power flow to the next program element through its rung output (ENO).



Power Flow Example 3

RLL Extensions to IEC 61131-3

This section describes the enhancements and other extensions to the IEC 61131-3 specification. InControl complies with IEC 61131-3 except where noted here.

Output Coils

You can place an Output Coil anywhere on a rung, including to the left of an input coil or within an OR branch. You can place multiple coils on a single rung, and the status of one coil is not affected by the status of the others. An Output coil stores the logical result of the logic evaluated up to its location on the rung.

OR Branches

You can insert an OR branch that contains no logic (a shunt). You can use a shunt to temporarily disable a section of logic without deleting it from the program. Used with a contact that turns off power flow to the logic in question, the shunt maintains power flow across the rest of the rung.

Counter Parameters

The Preset Value and the Current Value parameters used in the counter function blocks are DINT data types.

Math Function Blocks

The following math function blocks have inputs and outputs that accept any of the Any_Bit data types, except for the BOOL: ADD, DIV, SUB, MUL, MOD.

Unsupported Functions

InControl does not support the following functions, which are defined in the IEC 61131-3 specification: LIMIT, MUX, SEL

Unsupported Function Blocks

InControl does not support the following function blocks, which are defined in the IEC 61131-3 specification: SR, RS, SEMA, EDGE_CHECK, RTC.

Additional Built-In Functions and Function Blocks

InControl provides the following functions and function blocks, which are not defined in the IEC 61131-3 specification: ARRAY_TO_STRING, STRING_TO_ARRAY, CLOSEFILE, COPYFILE, DELETEFILE, NEWFILE, OPENFILE, READFILE, REWINDFILE, WRITEFILE, MSGWND, ABORT_ALL.

Creating an RLL Program

After starting InControl, you can create a new RLL program or edit an existing one.

To create a new RLL program:

1. On the **File** menu, click **New.**

   The menu of program types supported by InControl appears.

2. Select **RLL Program** and click **OK.** The **Save As** dialog box appears.
3. Choose a name (up to 31 characters) and directory (project) for the program and click **Save.** A new RLL program appears, showing the two power rails and one rung. The new program appears in the Project window.
4. Begin adding the program elements.

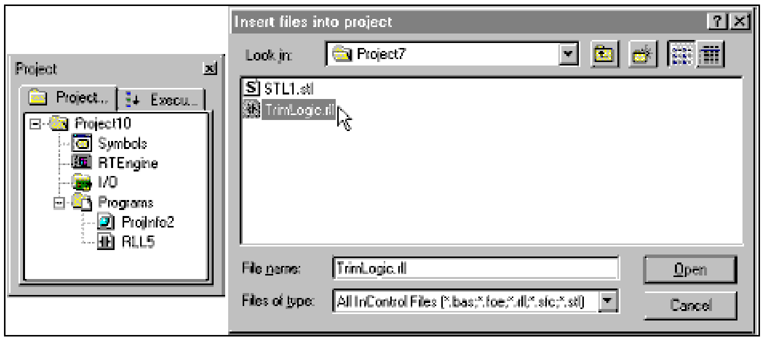
To edit an existing RLL program:

1. If the Project window is not open, click Project in the View menu. The Project window appears.
2. Double-click the name of the program to edit.

   The RLL editor opens, displaying the selected program.

You can also click **Open** in the **File** menu to open an existing program for editing. When the **Open** dialog box appears, select the program to open. If a program is not part of the current project, you can add it.

You can click **Files into Project** in the **Insert** menu to add any POU (program, function, function block, etc.) to a project. In the figure below, the program TrimLogic, shown in the **Insert Files into Project** dialog box, is selected and can be added to Project10. Note that the file itself is not copied or moved when it is added to another project.



Adding a POU to a Project

Note All POUs are inserted under the Programs folder of the Project window. You must move functions to the Functions folder, function block types to the Function Block folder, and macros to the Macros folder for the project to compile correctly.

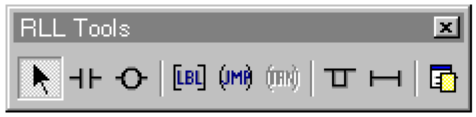
If you open a project developed under InControl 7.0, you have the option of converting the files to an InControl 7.1 project. Any macros in that project appear in the Programs folder after the conversion. You can move these macros to the Macros folder, but this is not required for the project to compile. Macros in the Rel. 7.0 project that have been excluded from download and that are called from another SFC will appear in the Macros folder after the conversion.

The RLL Tools

This section describes the RLL toolbar and gives some tips for editing a program.

Using the RLL Tool and Menu Bar

The RLL toolbar displays the tools used to create an RLL program.



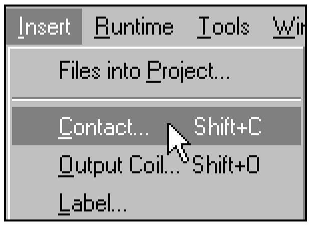
RLL Toolbar OPtions

Icon	Menu Bar Option	Function
↖	n/a	Allows you to select program elements.
┤├	Contact	Adds a contact to the program.
[LBL]	Label	Adds a label to the program.
-O-	Coil	Adds a coil to the program.
(JMP)	Jump Coil	Adds a jump coil to the program.
(TRN)	SFC Transition Coil	Adds an SFC transition coil to the program. Only available when an SFC program is being edited.
⊓	Branch	Adds an OR branch to the program.
├┤	Rung	Adds a new rung to the program.
⧉	n/a	Opens the Block palette.

Editing Tips

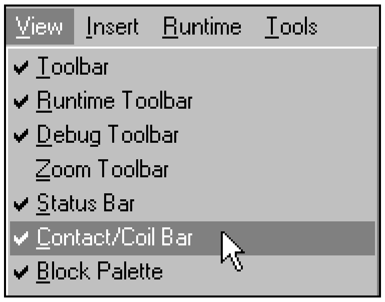
These tips can help as you edit a program.

- This chapter describes how to use these tools based on selections that you make from the RLL toolbar. You can also make tool choices from the **Insert** menu, which is shown in the following figure. To avoid confusion, only one method is described in this chapter.

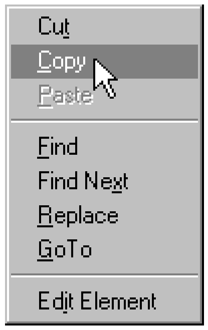
  When you insert a program element from the menu bar, the element is inserted at the current location of the cursor within the program. When you insert a program element using the toolbar, you can move the cursor to the location in the program where you want to place the element.

- Use the **View** menu to display those objects that you need to see during an editing session. For example, if you prefer to add program elements from the Menu bar, instead of the RLL toolbar, you can hide the RLL toolbar.

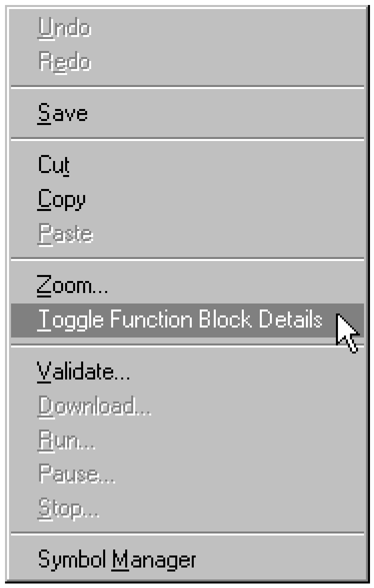
  

- During an editing session, you can right-click for a fast display of some of the editing options that appear in the menu bar.

  With the cursor over an RLL element, right-click to display the following menu:

  With the cursor in an editor window, but not over an RLL element, right-click to display the following menu:

  These menu options are described in the "InControl Environment" chapter.

- To print the RLL program, including configuration data for the function blocks, select **Function Block Details** on the **View** menu. Then print the program.

Adding Contacts

The contact typically represents a discrete input point, such as a limit switch. A contact can also represent an internal memory location, and as such, it is termed a Boolean. The contact can have one of two states: TRUE or FALSE. You refer to the contact in your program and the object that it represents by its symbolic name, which you assign in the Symbol Manager. For internal memory locations, you can assign the same symbolic name to a contact and a coil, and the output from one rung can serve as an input to another rung. InControl supports four types of contacts, which are described below.

Open Contact

The normally open contact operates as follows:

- The contact passes power flow if the point that it represents is TRUE.
- The contact does not pass power flow if the point that it represents is FALSE.

When you add a contact to a rung, you can use the reserved word TRUE or FALSE for the symbol name. The contact then operates as if it were always TRUE (on) or always FALSE (off). Do not add the contact as a local or global symbol.

Closed Contact

The normally closed contact operates as follows:

- The contact passes power flow if the point that it represents is FALSE.
- The contact does not pass power flow if the point that it represents is TRUE.

Positive Transition Contact

The positive transition sensing contact operates as follows:

- When the status of the contact is evaluated, the current state of the point it represents is compared to its state in the previous scan. If the point is TRUE in the current scan, but was FALSE in the previous scan, the contact passes power flow. Otherwise, the contact does not pass power flow. If the point transitions from FALSE to TRUE and back to FALSE again before the contact is reevaluated, the contact does not pass power flow.
- The contact cannot pass power flow again until the point that it represents transitions from FALSE to TRUE again.

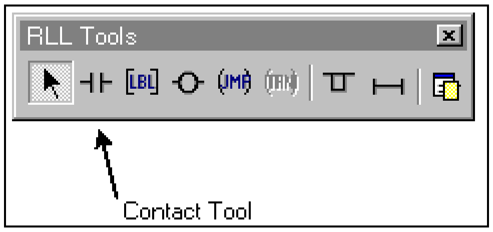
Negative Transition Contact

The negative transition sensing contact operates as follows:

- When the status of the contact is evaluated, the current state of the point it represents is compared to its state in the previous scan. If the point is FALSE in the current scan, but was TRUE in the previous scan, the contact passes power flow. Otherwise, the contact does not pass power flow. If the point transitions from TRUE to FALSE and back to TRUE again before the contact is reevaluated, the contact does not pass power flow.
- The contact cannot pass power flow again until the point that it represents transitions from TRUE to FALSE again.

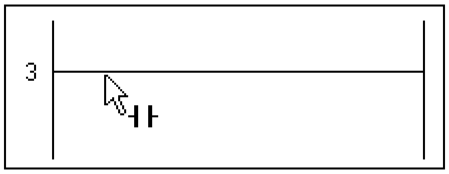
To add a contact to the program:

1. Click the **Contact Tool** on the RLL toolbar.

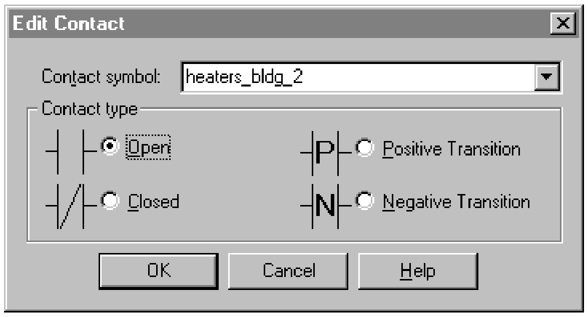


   The cursor changes into the contact cursor.

2. Move the cursor to the location on the rung where you want to place the new contact.



3. Click the left mouse button. The Edit Contact dialog box appears.



Contact symbol: heaters_bldg_2

Contact type: [✓ Open] Closed Positive Transition Negative Transition

   Edit Contact Dialog Box

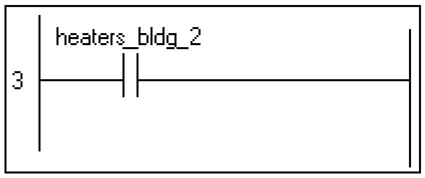
4. If you have already defined the variable names for your system, select the variable name to represent the contact (heaters_bldg_2 in the figure).

   If you have not defined a variable name for this contact, enter a name in the **Contact Symbol** field

5. Select the contact type (Open, Closed, etc.) and click **OK.**

   If the variable is new, the system prompts you to, click **Add Local** or **Add Global** to add the new variable name to the Symbol Manager as a local or global variable.

   If you click **OK** without adding the symbol to the Symbol Manager, the editor accepts the name, but you must still add the variable to the Symbol Manager before the program can compile.



Note You can create a contact that is always TRUE or always FALSE. Enter either TRUE or FALSE in the **Contact Symbol** field and then click **OK.** The contact then operates as if it were forced TRUE (on) or FALSE (off).

Adding Coils

The coil typically represents a discrete output point, such as a solenoid. A coil can also represent an internal memory location, and as such, it is termed a Boolean. You refer to the coil in your program and the object that it represents by its symbolic name, which you assign in the Symbol Manager. For internal memory locations, you can assign the same symbolic name to a contact and a coil, and the output from one rung can serve as an input to another rung. As an enhancement to IEC 61131-3, you can place multiple coils on a single rung, and the status of one coil is not affected by the status of the others. A coil stores the partial Boolean result evaluated to its point on a rung.

InControl supports six types of coils, which are described in the pages that follow.

Output Coil

The output coil operates as follows:

- The coil sets the point that it represents to TRUE when the coil has power flow.
- The coil sets the point that it represents to FALSE if the coil does not have power flow.

Negated Output Coil

The negated output coil operates as follows:

- The coil sets the point that it represents to TRUE when the coil does not have power flow.
- The coil sets the point that it represents to FALSE if the coil has power flow.

Set (Latch) Coil

The set (latch) coil operates as follows:

- The coil sets the point that it represents to TRUE when the coil has power flow.
- The point continues to be TRUE (the point is set, or latched) even when the coil no longer has power flow.
- The point can be set to FALSE by a reset coil.

Reset (Unlatch) Coil

The reset (unlatch) coil operates as follows:

- The coil sets the point that it represents to FALSE when the coil has power flow.
- The point remains FALSE (the point is reset, or unlatched) even when the coil no longer has power flow.
- The point can be set to TRUE by a set coil.

Positive Transition Coil

The positive transition sensing coil operates as follows:

- When the status of the coil is evaluated, the state of the power flow into the coil during the current scan is compared to its state in the previous scan. If the power flow is TRUE in the current scan, but was FALSE in the previous scan, the coil pulses, setting the point it represents to TRUE.
- The point remains TRUE, unless it is set to FALSE, for the duration of the scan of the ladder logic.
- After the coil has pulsed, power flow must be FALSE for at least one scan before the coil can pulse again.

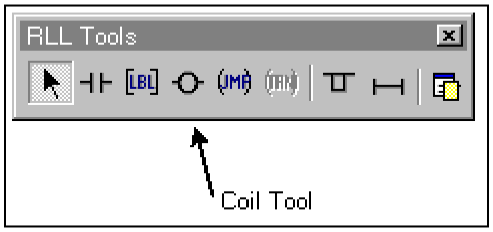
Negative Transition Coil

The negative transition sensing coil operates as follows:

- When the status of the coil is evaluated, the state of the power flow into the coil during the current scan is compared to its state in the previous scan. If the power flow is FALSE in the current scan, but was TRUE in the previous scan, the coil pulses, setting the point it represents to TRUE.
- The point remains TRUE, unless it is set to FALSE, for the duration of the scan of the ladder logic.
- After the coil has pulsed, power flow must be TRUE for at least one scan before the coil can pulse again.

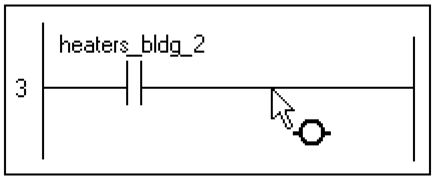
To add a coil to the program:

1. Click the **Coil Tool** on the RLL toolbar.

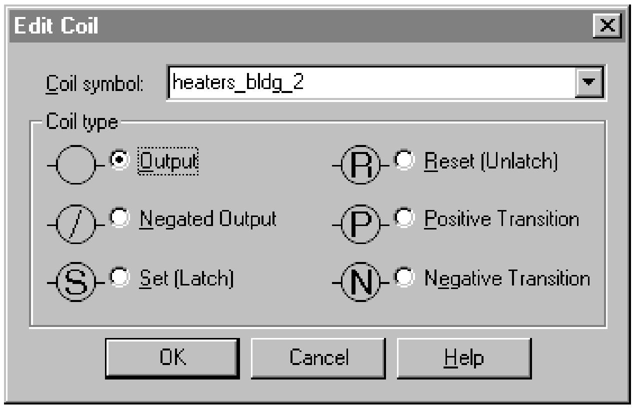


The cursor changes into the coil cursor.

2. Move the cursor to the location on the rung where you want to place the new coil.



3. Click the left mouse button. The Edit Coil dialog box appears.



Edit Coil

Coil symbol: heaters_bldg_2

Coil type

[✓ Output] Reset (Unlatch)

Negated Output Positive Transition

Set (Latch) Negative Transition

OK Cancel Help

Edit Coil Dialog Box

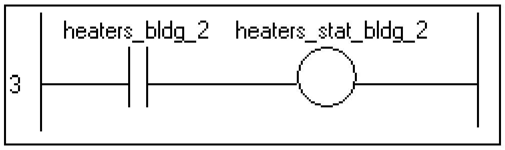
4. If you have already defined the variable names for your system, click the variable name to represent the coil (heaters_stat_bldg_2 in the figure).

   If you have not defined a variable name for this coil, enter a name in the **Coil Symbol** field.

5. Select the coil type (Output, Negated Output, etc.) and click **OK.**

   If the variable is new, the system prompts you to, click **Add Local** or **Add Global** to add the new variable name to the Symbol Manager as a local or global variable.

   If you click **OK** without adding the symbol to the Symbol Manager, the editor accepts the name, but you must still add the variable to the Symbol Manager before the program can compile.

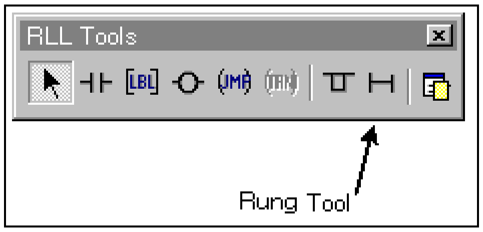


You can place an Output Coil anywhere on a rung, including to the left of an input coil or within an OR branch. An Output coil stores the logical result of the logic evaluated up to its location on the rung.

Adding Rungs

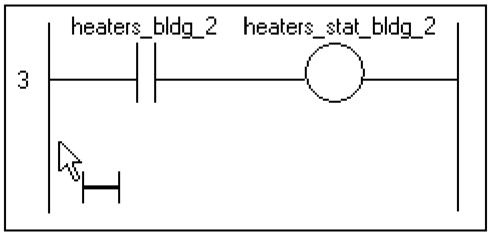
To add a rung to the program:

1. Click the **Rung Tool** on the RLL toolbar.



   The cursor changes into the Rung Tool cursor.

2. Move the cursor to the location on the left power rail where you want to insert the new rung.

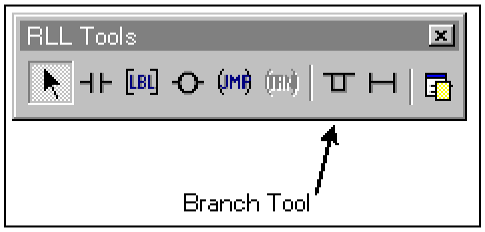


3. Click the left mouse button. The editor inserts the rung at the specified location.

Adding OR Branches

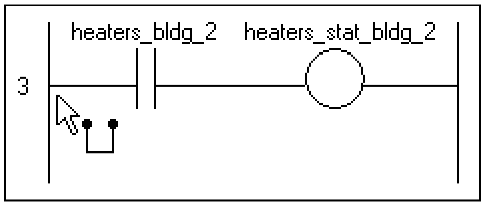
To add an OR branch to a rung:

1. Click the OR Branch Tool on the RLL toolbar.

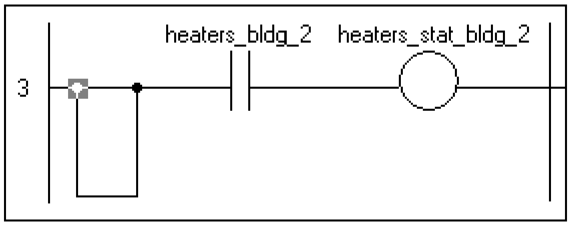


   The cursor changes into the OR Branch Tool cursor.

2. Move the cursor to the location on the rung where you want to insert the OR branch.



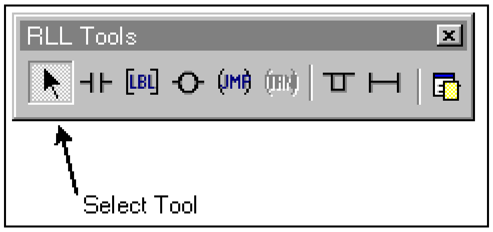
3. Click the left mouse button. The editor inserts the OR at the specified location.



After inserting the OR branch, you can adjust the contact points as needed.

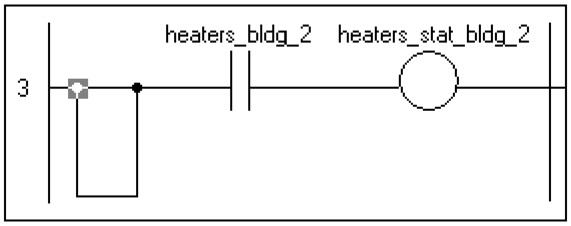
To move the contact points of an OR branch:

1. Click the Select Tool.

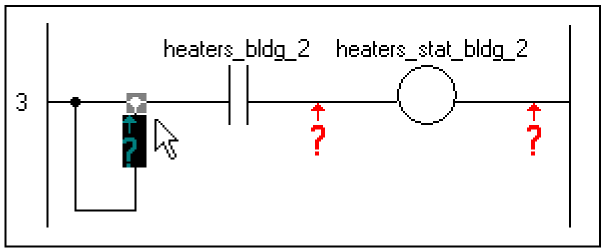


2. Click the contact point that you want to move.

3. Drag the contact point to the new location on the rung. The editor connects the OR branch at the new location on the rung.



To determine where you can move a contact point, place the cursor over the contact point and double-click. A series of question marks appear to show valid locations on the rung, as shown in the following figure. This can be very helpful when OR branches are nested.



Valid Connection Points

Click one of the question marks to return to the normal edit mode. The contact point that is selected moves to the location of the selected question mark.

As an enhancement to IEC 61131-3, you can insert an OR branch that contains no logic (a shunt). You can use a shunt to temporarily disable a section of logic without deleting it from the program. Used with a contact that turns off power flow to the logic in question, the shunt maintains power flow across the rest of the rung. This feature is useful for debugging your program. You can also move a contact point from rung to rung without deleting the logic contained within the OR branch.

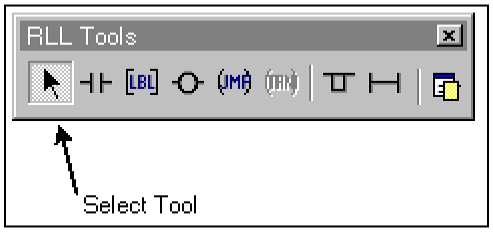
Note If you are editing an RLL rung contained within an SFC RLL Transition, you cannot add a second rung. For more information about SFCs, see the "SFC Program Elements" chapter.

Deleting OR Branches

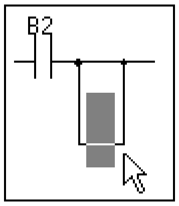
The operation of the **Cut Tool** is based on your selecting an object and then clicking on the **Cut Tool** to delete the object. To delete the OR branch, however, follow one of these procedures.

To delete an OR Branch that contains no elements:

1. Click the Select Tool.



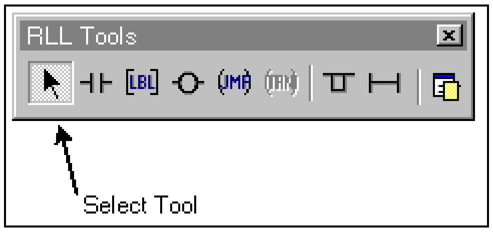
2. Place the cursor in the middle of the OR branch and click.



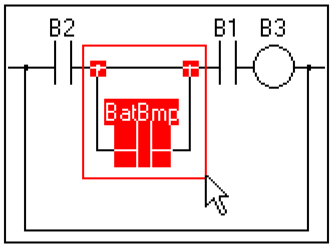
3. Click the Cut Tool to delete the OR branch.

To delete an OR Branch that contains one or more elements:

1. Click the Select Tool.



2. Drag an area that includes the entire OR branch and its connection points.



Selecting the OR Branch

3. Click the **Cut Tool** to delete the OR branch.

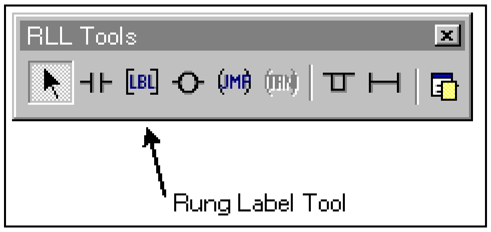
Adding Labels and Jump Coils

Use the jump coil and label elements to disable sections of program code temporarily. You must use the jump coil and label together. A jump coil without a label causes an error when you compile the program. The jump coil and label operate as follows:

- When a jump receives power flow, program execution ignores all logic between the jump and its corresponding label.
- When a jump coil is actively skipping logic, outputs between the jump coil and the label are not activated.
- All logic between a jump coil and label is executed normally when the jump coil does not receive power flow.
- Program execution cannot jump backwards to a previous rung.

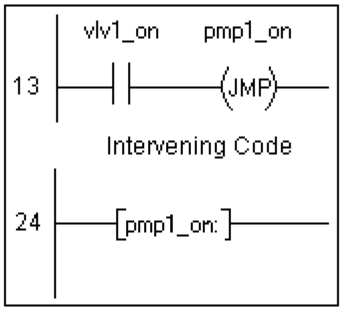
To add a label to the program:

1. Click the **Label Tool** on the RLL toolbar.

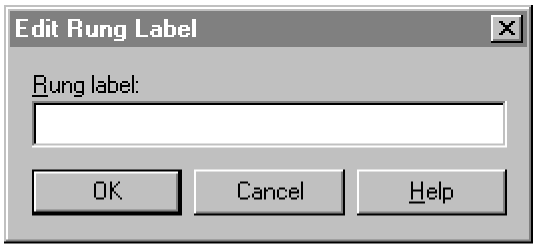


The cursor changes into the Label cursor.

2. Move the cursor to the location on the rung where you want to place the new label.

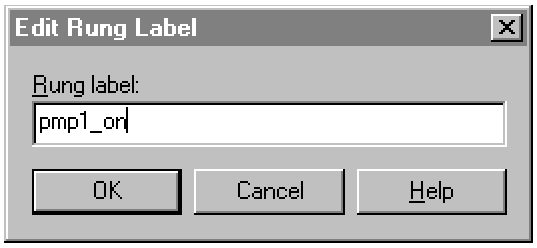


3. Click the left mouse button. The **Edit Rung Label** dialog box appears.

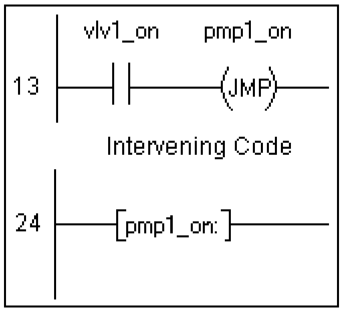


Edit Rung Label Dialog Box

4. Enter a label and click **OK.** A label name cannot contain any spaces.



The editor inserts the label at the specified location.

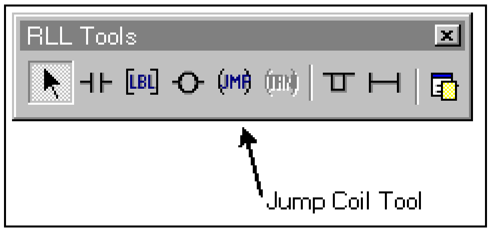


Example Label

You must use the jump coil and label together. A jump coil without a label causes an error when you compile the program.

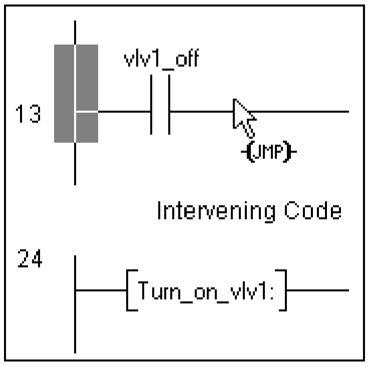
To add a jump coil to the program:

1. Click the **Jump Coil Tool** on the RLL toolbar.

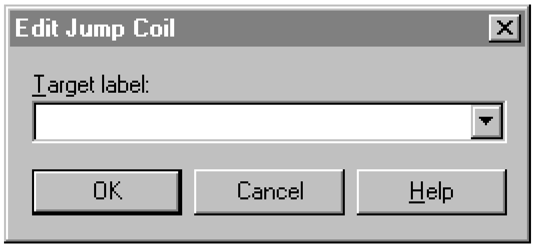


The cursor changes into the Jump Coil cursor.

2. Move the cursor to the rung where you want to place the new jump coil.

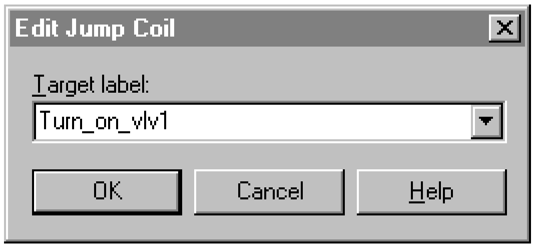


3. Click the left mouse button. The **Edit Jump Coil** dialog box appears.



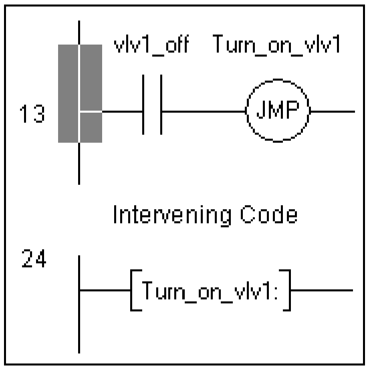
Edit Jump Coil Dialog Box

4. Enter a target label and click **OK.** A target label name cannot contain any spaces.



Target label: Turn_on_vlv1

The editor inserts the jump coil at the specified location.



Example Jump Coil

Adding SFC Transition Coils

The SFC transition coil is an RLL program element that you can use only under specific conditions: in an SFC program, within an Action that is associated with a Step or a Macro Step. The SFC transition coil, which executes similarly to the jump coil element, operates as described below.

For more information about using SFCs and Actions, refer to "Action" in the "SFC Program Elements" chapter.

Using Transition Coils in Non-Stored Actions

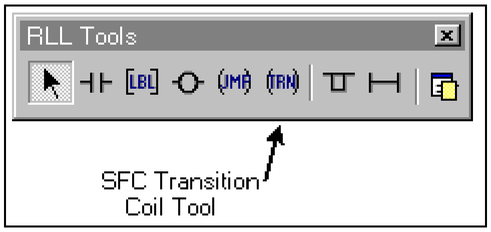
- When a transition coil associated with a Step receives power flow, the remaining code within the Step is aborted. Program execution then jumps to the Label in the SFC that is specified in the SFC transition coil. No Steps are stopped other than the associated Step.
- When a transition coil associated with a Macro Step receives power flow, the child SFC is aborted. Program execution then jumps to the Label in the parent SFC that is specified in the SFC transition coil.

Using Transition Coils in Stored Actions

- When a transition coil associated with a Step or Macro Step receives power flow, all active Steps in the SFC, including those in all child SFCs, are aborted. On the next scan, program execution then jumps to the Label in the parent SFC that is specified in the SFC transition coil.
- If the stored Action is located within a Macro, all active Steps, including those in any child SFCs are aborted. However, Steps in the parent SFC, including the Macro Step, are not affected.

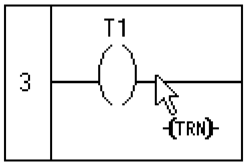
To add an SFC transition coil to the program, you must be editing an Action in an SFC program. With an Action open for editing, follow these steps.

1. Click the **SFC Transition Coil** tool on the RLL toolbar.

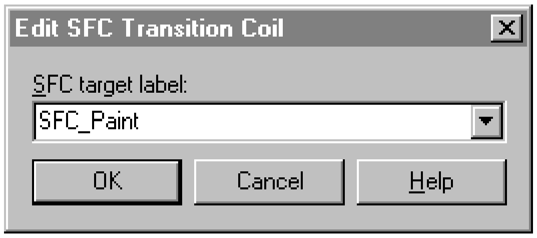


The cursor changes into the SFC Transition Coil cursor.

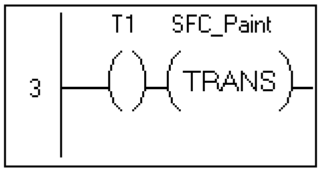
2. Move the cursor to the location on the rung where you want to place the SFC Transition Coil.



3. Click the left mouse button. The **Edit SFC Transition Coil** dialog box appears.



4. Enter the name of the SFC target and click **OK.** An SFC target label name cannot contain any spaces. The editor inserts the SFC Transition Coil at the specified location.



Adding Functions / Function Blocks

InControl RLL programs support predefined and user-defined functions and function blocks.

Predefined Functions / Function Blocks

Several predefined functions and function blocks are available for you to use in an RLL program. You can enable functions and function blocks with inputs from an RLL rung, have them do operations such as trigonometric, math, logic functions, bit shift operations, file operations, etc., and then send the results to an output that feeds into another element on the RLL rung.

The predefined functions and function blocks supported by InControl are listed in the following table. For information about syntax and operation, see the *InControl Function and Function Block Reference Manual.*

Functions/Procedures by Group

Group	Type	Description
Bitwise	AND	Computes the bitwise AND of two numbers.
	NOT	Computes the bitwise complement of a number.
	OR	Computes the bitwise OR of two numbers.
	ROL	Rotates the input left by a specified number of bits.
	ROR	Rotates the input right by a specified number of bits.
	SHL	Shifts the input left by a specified number of bits.
	SHR	Shifts the input right by a specified number of bits
	XOR	Computes the bitwise Exclusive OR of two numbers.
Comparison	EQ	Tests two inputs for equality.
	GE	Tests if first input is greater than or equal second input.
	GT	Tests if first input is greater than second input.
	LE	Tests if first input is less than or equal second input.
	LT	Tests if first input is less than second input.
	NE	Tests two inputs for inequality.

Group	Type	Description
Conversion	ARRAY_TO_ STRING	Takes a byte array input and stores the bytes as characters in a string.
	BCD_TO_INT	Converts a Binary-Coded Decimal (BCD) input to an ANY_INT value.
	DATE_TO_ REAL	Converts a DATE data type input to an ANY_REAL value.
	DATE_TO_ STRING	Converts a DATE data type input to a string.
	INT_TO_BCD	Converts an integer to the equivalent Binary-Coded Decimal (BCD) representation of the value.
	INT_TO_REAL	Converts an ANY_INT input to an ANY_REAL value.
	INT_TO_ STRING	Converts an ANY_INT input to a string.
	REAL_TO_ DATE	Converts an ANY_REAL input to a DATEvalue.
	REAL_TO_INT	Converts an ANY_REAL input to an ANY_INT value.
	REAL_TO_ STRING	Converts an ANY_REAL input to a string.
	REAL_TO_ TIME	Converts an ANY_REAL input to a TIMEvalue.
	STRING_TO_ ARRAY	Takes a string input and stores the characters of the string in a byte array.
	STRING_TO_ DATE	Converts an input string to a DATE value.
	STRING_TO_ INT	Converts an input string to an ANY_INT value.
	STRING_TO_ REAL	Converts a string input to an ANY_REAL value.
	STRING_TO_ TIME	Converts a string input to a TIME value.
	TIME_TO_ REAL	Converts a TIME input to an ANY_REAL value.
	TIME_TO_ STRING	Converts a TIME input to a string.
Counter	CTD	Counts events by decrementing by one.
	CTU	Counts events by incrementing by one.
	CTUD	Counts events up or down.

Group	Type	Description
File	CLOSEFILE	Closes a file.
	COPYFILE	Copies a file.
	DELETEFILE	Deletes a file.
	NEWFILE	Creates a new file.
	OPENFILE	Opens an existing file.
	READFILE	Reads data from a file.
	REWINDFILE	Rewinds a file to the beginning.
	WRITEFILE	Writes data to a file.
Math	ABS	Computes the absolute value of a value.
	ADD	Adds two values.
	DIV	Divides one value by another.
	EXPT	Raises a value to the power specified by a second value.
	MAX	Determines the larger of two values.
	MIN	Determines the smaller of two values.
	MOD	Divides one value by another and stores the remainder.
	MOVE	Copies data from one location to another.
	MUL	Multiplies two values.
	NEG	Negates (inverts) the inputs.
	SQRT	Computes the square root of a value.
	SUB	Subtracts one value from another.
	TRUNC	Removes one or more of the least significant digits of an ANY_REAL data type.
String	CONCAT	Concatenates a string input to the end of another string.
	DELETE	Deletes characters from the middle of a string input.
	FIND	Searches for one string input within another.
	INSERT	Inserts a string input into another string.
	LEFT	Copies the leftmost characters from a string input.
	LEN	Stores the length of a string input.
	MID	Copies characters from the middle of a string input.
	MSGWND	Displays a message in the Output Window.
	REPLACE	Replaces characters in a string input with another string input.
	RIGHT	Copies the rightmost characters from a string input.

Group	Type	Description
Timer	TOF	Provides off-delay timing of events.
	TON	Provides on-delay timing of events.
	TP	Activated by a pulse, provides off-delay timing of events.
Trig/Log	ACOS	Computes the arc cosine of a value.
	ASIN	Computes the arc sine of a value.
	ATAN	Computes the arc tangent of a value.
	COS	Computes the cosine of a value.
	EXP	Computes the natural log exponentiation of a value.
	LN	Computes the natural log of a value.
	LOG	Computes the log (base 10) of a value.
	SIN	Computes the sine of a value.
	TAN	Computes the tangent of a value.
Trigger	ABORT_ALL	Aborts all programs that are running.
	F_TRIG	Turns on an output when triggered by a falling edge trigger.
	R_TRIG	Turns on an output when triggered by a rising edge trigger.

User-Defined Functions / Function Blocks

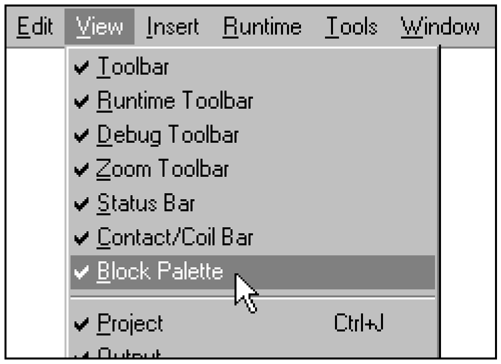
You can develop a user-defined function or function block in RLL code and call it from any type of program, RLL, STL, etc. For an example that shows how to develop a user-defined function, see the "RLL Example Program Appendix."

Note these guidelines when you develop a function or function block.

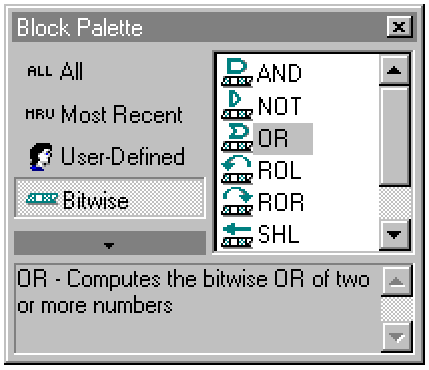
- You can define up to seven input or input-output (InOut) parameters. The editor always adds an eighth input by default, which acts as the EN input.
- You can define up to seven output parameters. The editor always adds an eighth output by default, which acts as the ENO output.
- If you define a Boolean output parameter, the state of this output determines the output state of the rung, which contains the function, in the calling program.

To add a function or function block to the RLL program:

1. If the Block palette is not being displayed, click **Block Palette** in the **View** menu.



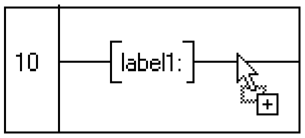
The editor displays the Block Palette.



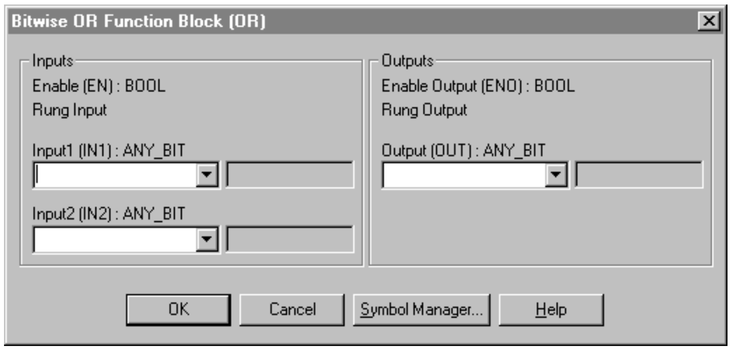
OR - Computes the bitwise OR of two or more numbers

2. Select the specific function or function block that you want to add, such as an OR Bitwise block.

3. Drag the function/function block to the rung.



The dialog box for the block appears. The dialog box for the OR Function is shown in the following figure as an example.

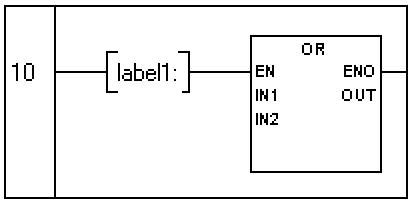

Bitwise OR Function Block (OR)

Inputs
Enable (EN) : BOOL
Rung Input
Input1 (IN1) : ANY_BIT
Input2 (IN2) : ANY_BIT

Outputs
Enable Output (ENO) : BOOL
Rung Output
Output (OUT) : ANY_BIT

Example Function Block Dialog Box

4. Fill in the appropriate information for the function or function block.
5. When you have finished filling out the dialog box, click **OK.** The editor inserts the block at the specified location.



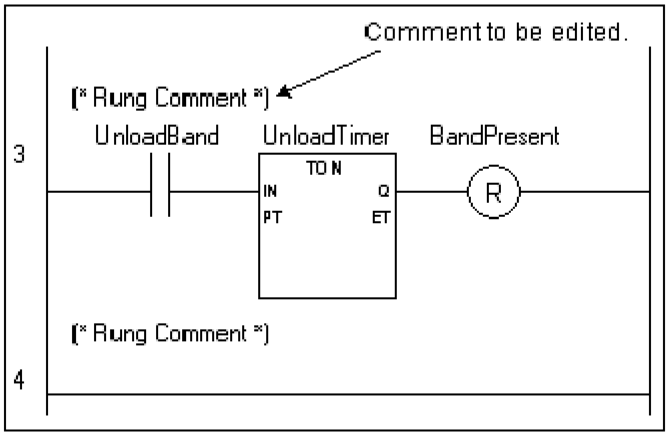
Adding a Comment

You can enter a descriptive comment for each rung in the program. Comments can be several lines if necessary. They are not downloaded to the runtime engine.

To enter a comment for a rung:

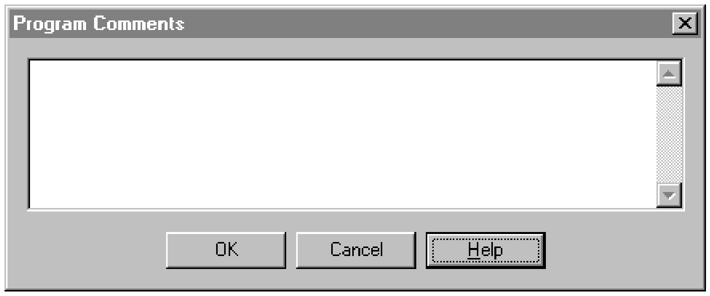
1. Open an RLL program.

   If the **Program Comments** option has been selected on the **View** menu, the text "Rung Comment" appears between every rung.

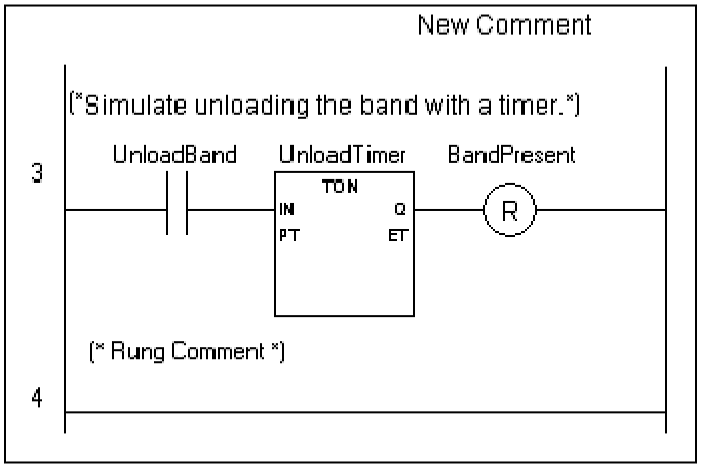


   Entering a Comment Example 1

2. If the text "Rung Comment" is not visible, click **Program Comments** on the **View** menu. The text "Rung Comment" appears between every rung in the program that does not already have a comment.
3. Double-click the "Rung Comment" that you want to edit. The Program Comments dialog box appears.



4. Enter the comment and click OK. Your comment appears within the program.



Entering a Comment Example 2

C H A P T E R 2

Using the SFC Editor

This chapter introduces the Sequential Function Chart (SFC) editor and how to use it to create a new program and to add elements (Steps, Transitions, Jumps, Macro Steps, etc.) to it. For details about the elements themselves, refer to the "SFC Program Elements" chapter.

Contents

Creating an SFC Program

After starting InControl, you can create a new SFC program or edit an existing one.

To create a new SFC program:

1. On the **File** menu, click **New.**

   The menu of program types supported by InControl appears.
2. Select **SFC Program** and click **OK.** The **Save As** dialog box appears.
3. Choose a name (up to 31 characters) and directory (project) for the program and click **Save.** A new SFC program appears, showing a starting Step and an end Step.
4. To add program elements, see "Adding Program Elements."

To edit an existing SFC program:

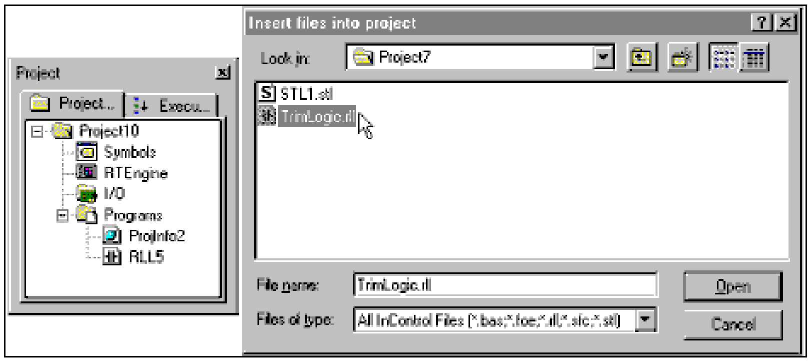
1. If the Project window is not open, click **Project** on the **View** menu. The Project window appears.
2. Double-click the program that you want to edit.

   The SFC editor opens, displaying the selected program.

You can also click **Open** on the **File** menu to open an existing program for editing.

When the **Open** dialog box appears, select the program to open. If a program is not part of the current project, you can add it.

You can click **Files into Project** on the **Insert** menu to add any POU (program, function, function block, etc.) to a project. In the figure below, the SeamWeld SFC program, shown in the **Insert Files into Project** dialog box, is selected and can be added to Project 55. Note that the file itself is not copied or moved when it is added to another project.



Adding an SFC POU to a Project

Note All POUs are inserted under the Programs folder of the Project window. You must move functions to the Functions folder, function block types to the Function Block folder, and macros to the Macros folder for the project to compile correctly.

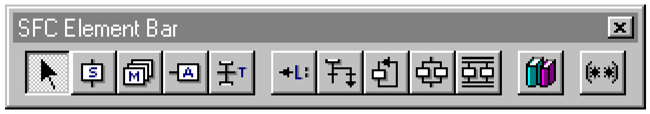
If you open a project developed under InControl 7.0, you have the option of converting the files to an InControl 7.1 project. Any macros in that project appear in the Programs folder after the conversion. You can move these macros to the Macros folder, but this is not required for the project to compile. Macros in the Rel. 7.0 project that have been excluded from download and that are called from another SFC will appear in the Macros folder after the conversion.

The SFC Tools

This section describes the SFC toolbar and gives some tips for editing a program.

Using the SFC Tool and Menu Bar

The SFC toolbar displays the tools used to create an SFC program.



The SFC toolbar options are described in the following table.

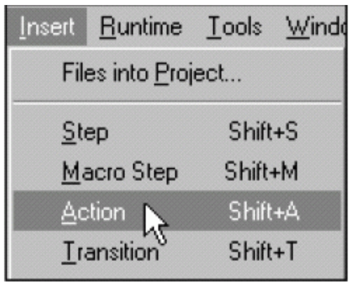
Icon	Menu bar Option	Function
	n/a	Allows you to select program elements.
	Insert/Step	Adds a Step to the program.
	Insert/Macro Step	Adds a Macro Step to the program.
	Insert/Action	Adds an Action to the program.
o	Insert/Transitio n	Adds a Transition to the program.
	Insert/Label	Adds a Label to the program.
	Insert/Jump	Adds a Jump to the program.
	Insert/Loop	Adds a Loop to the program.

Icon	Menu bar Option	Function
▣	Insert/Select Diverge	Adds a Select Divergence to the program.
▣	Insert/Parallel Diverge	Adds a Parallel Divergence to the program.
▣	Insert/Library Step	Adds a predefined program Step from the library to the program.
(**)	Insert/Comment	Allows you to add program comments to the program.

Editing Tips

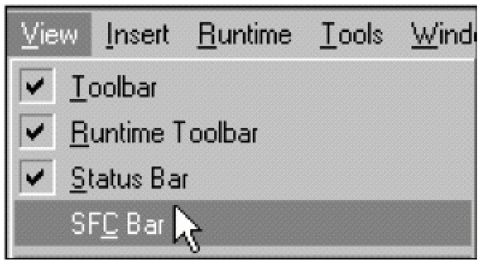
These tips can help as you edit a program.

- This chapter describes how to use these tools based on selections that you make from the SFC toolbar. You can also make tool choices from the **Insert** menu, which is shown below. To avoid confusion, only one method is described in this chapter.



When you insert a program element from the menu bar, the element is inserted at the current location of the cursor within the program. When you insert a program element using the toolbar, you can move the cursor to the location in the program where you want to place the element.

- Use the **View** menu to display those objects that you need to see during an editing session. For example, if you prefer to add program elements from the menu bar, instead of the SFC toolbar, you can hide the SFC toolbar.



- During an editing session, you can right-click in the SFC editor window for a fast display of some of the edit options that appear in the menu bar.
- To print the code used in SFC Steps, select **Structured Text** in the **Edit Step** dialog box for the appropriate Steps. Then print the program.

When you enter code used in SFC Steps, you use the same tools and menu options that are in the STL editor. For more information about the STL editor, see "Using the Structured Text Editor" chapter.

Adding Program Elements

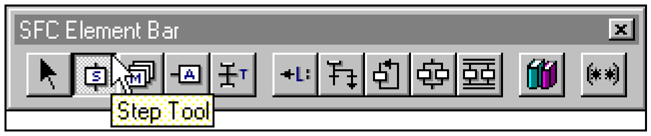
This section describes how to add SFC program elements using tools on the SFC toolbar.

Adding a Step

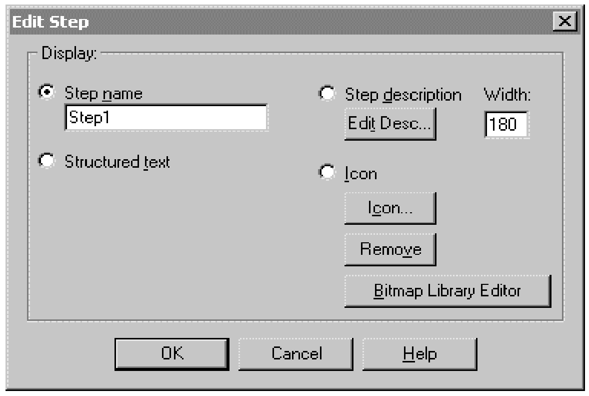
A Step represents a condition in which the behavior of the system follows a set of rules defined by the Actions and functions associated with the Step. See "Step" in the "SFC Program Elements" chapter for detailed information about using steps.

To add a Step to the program:

1. Click the **Step Tool** on the SFC toolbar. The cursor changes into the Step cursor.



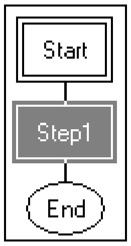
2. Move the cursor to the location in the program where you want to place the new Step and click. The **Edit Step** dialog box appears.



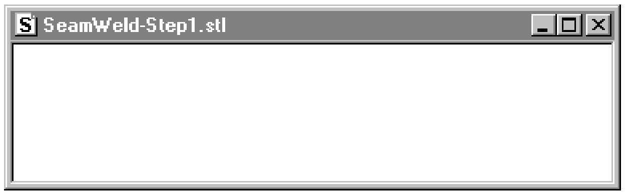
Edit Step Dialog Box

3. Enter the appropriate information for displaying the Step in the SFC as described in "Step" in the "SFC Program Elements" chapter. Then click **OK.**

   The new Step appears in the program at the location you specified.



4. To enter the Structured Text code for the Step, double-click the Step. A Structured Text editor window appears.



5. Enter the program code, described in the "Structured Text Language" chapter.

Note You can also add a Step from the Step library. A Step Template contains a user-defined code template that does a specific function and an icon that is appropriate for the function. See "Adding a Library Step."

InControl allows you to protect program code within an SFC Step from unauthorized changes. Select the **Lock Algorithms** command in the **Edit** menu and assign a password. To lock the SFC code, you must have access to the Edit Program security task.

For more information about locking SFC algorithms, see "Locking SFC Algorithms" in the "Setting Up Security" chapter.

Adding a Transition

A Transition represents the condition that causes control to pass from one or more Steps preceding the Transition to one or more successive Steps that follow the Transition. See "Transition" the "SFC Program Elements" chapter for detailed information about using Transitions in the program.

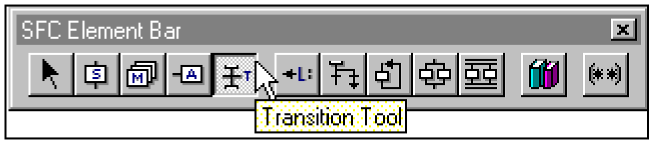
You can choose from two types of Transitions in an SFC program.

- The RLL Transition is based on RLL code.
- The Boolean Transition is based on Boolean logic and Structured Text Boolean expressions.

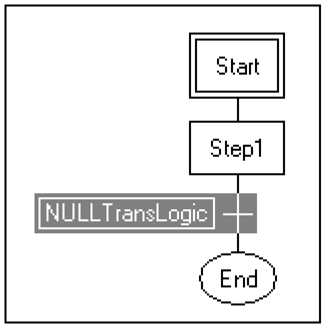
Adding RLL Transitions

To add an RLL Transition to the program:

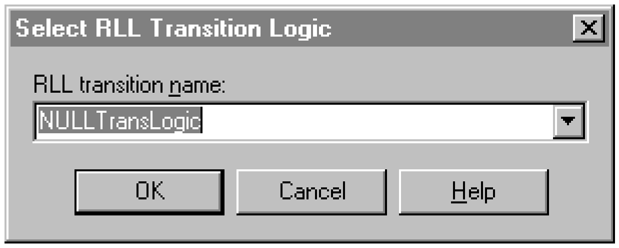
1. If **Boolean Transition** on the **Edit** menu is checked, click it to deselect that option.
2. Click the **Transition Tool** on the SFC toolbar. The cursor changes into the Transition cursor.



3. Move the cursor to the location in the program where you want to place the new Transition and click. The new Transition appears in the program.



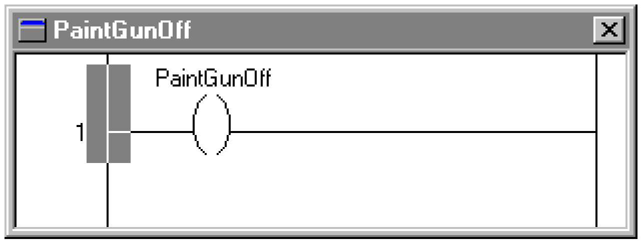
4. To enter the RLL code, double-click the Transition. The **Select RLL Transition Logic** dialog box appears.



Select RLL Transition Dialog Box

5. Enter a meaningful name for the Transition.

   An RLL editor window appears containing an RLL rung.



6. Add the RLL code, described in the "RLL Program Elements" chapter. Note that an SFC Transition can have only one RLL rung.
7. Close the editor window.

To delete an existing RLL transition permanently:

1. Select the Transition in the SFC and click the **Cut tool** to delete the Transition.
2. On the **Tools** menu, click **RLL Transition Manager.** The **RLL Transition Manager** dialog box appears.
3. Select the Transition to delete and click the **Delete** button. The Transition is removed from the Transition Manager.

Note You must delete the Transition from both the RLL Transition Manager and the SFC to remove the Transition code completely from your program.

To rename an existing RLL Transition:

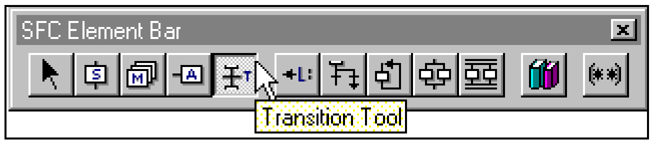
1. On the Tools menu, click RLL Transition Manager. The RLL Transition Manager dialog box appears.
2. Select the Transition to rename and click the Rename button. The Transition is renamed.

Note You must rename individual RLL Transitions after making a name change in the RLL Transition Manager.

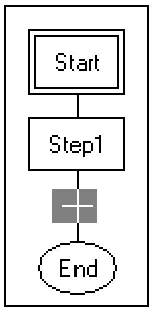
Adding Boolean Transitions

To add a Boolean Transition to the program:

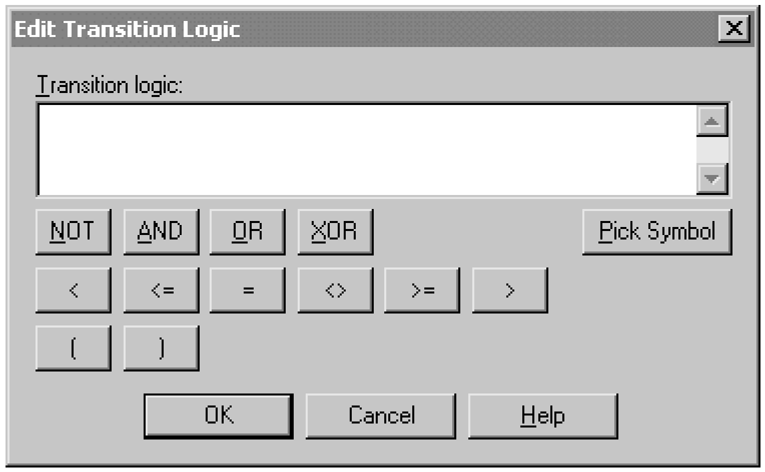
1. If **Boolean Transition** on the **Edit** menu is not checked, click to select it.
2. Click the **Transition Tool** on the SFC toolbar. The cursor changes into the Transition cursor.



3. Move the cursor to the location in the program where you want to place the new Transition and click. The new Transition appears in the program.



4. To enter the Boolean code, double-click the Transition. The **Edit Transition Logic** dialog box appears.



Edit Transition Logic Dialog Box

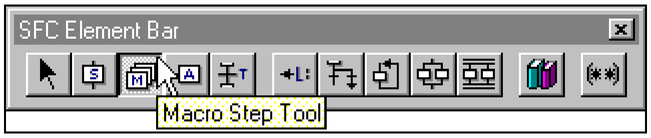
5. To enter the Boolean code for the Transition, type the code in directly or click **Pick Symbol** and select it from the Symbol Manager.
6. Click **OK** to save your work and close the dialog box.

Adding a Macro Step

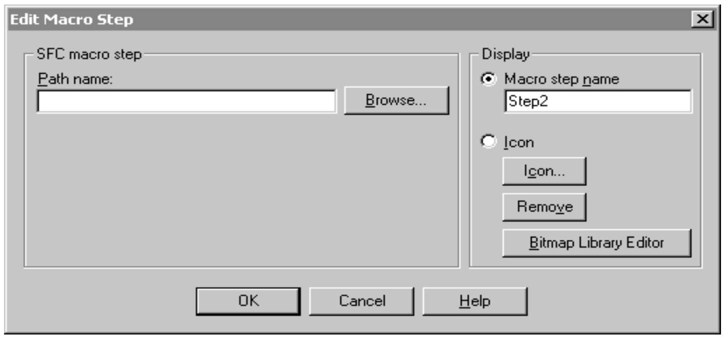
A Macro Step provides a means of calling another SFC from the currently executing SFC. See "Macro Step" in the "SFC Program Elements" chapter for detailed information about using Macro Steps in the program.

To add a Macro Step to the program:

1. Click the **Macro Step Tool** on the SFC menu bar. The cursor changes into the Macro Step Tool cursor.



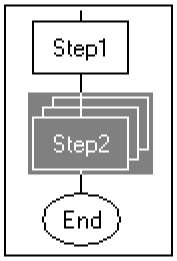
2. Move the cursor to the location in the program where you want to place the new Macro Step and click. The **Edit Macro Step** dialog box appears.



Edit Macro Step Dialog Box

3. Enter the information for displaying the Macro Step and for selecting the macro SFC, described in the "SFC Program Elements" chapter. Then click **OK.**

   The new Macro Step appears in the program.



4. Double-click the Macro Step to edit the macro SFC.

   If you right-click the Macro Step and select **Step Properties** you can change the Macro Step name or how it is displayed in the SFC.

Adding an Action

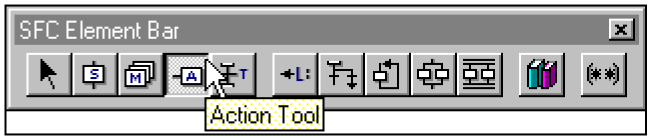
An Action consists of one or more sections of RLL code that are associated with a Step or a Macro Step. The system executes an Action when its associated Step becomes active. See "Action" in the "SFC Program Elements" chapter for detailed information about Actions.

You must create a Step or Macro Step before adding an Action to a program.

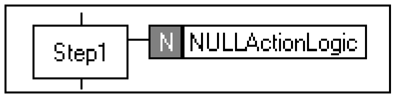
Adding New Actions

To add an Action to the program:

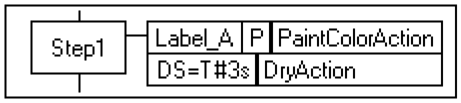
1. Click **Action Tool** on the SFC toolbar. The cursor becomes the Action Tool cursor.



2. Move the cursor to the location in the program (either on top of a Step or on top of a Macro Step) where you want to place the new Action and click. The new Action appears in the program.



3. You can add more than one Action to a Step or Macro Step by placing the cursor on top of an existing Action.



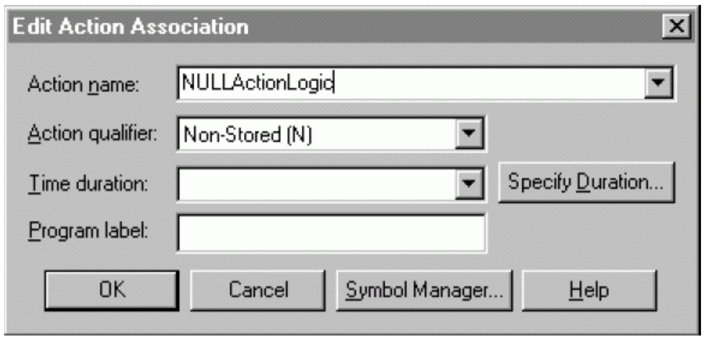
Step with Multiple Actions

Note To copy an Action to another Step in an SFC, hold down the **Ctrl** key and then drag the Action to the destination Step.

Editing New Actions

To edit a new Action:

1. Double-click the Action. The **Edit Action Association** dialog box appears.



Action name: NULLActionLogic

Action qualifier: Non-Stored (N)

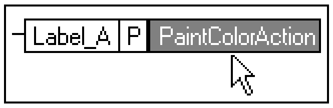
Edit Action Association Dialog Box

2. Enter the information for configuring the Action, described in "Action" in the "SFC Program Elements" chapter. Then click **OK** to save your changes. The system closes the dialog box and then displays an empty rung of RLL ready for editing.
3. Enter the RLL code, described in the "RLL Program Elements" chapter.

Editing Existing Actions

To edit the RLL of an existing Action:

1. Double-click the right side of the Action as shown below.



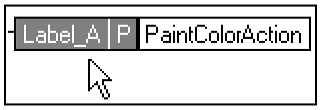
The RLL code for the Action appears.

2. Enter the RLL code as described the "RLL Program Elements" chapter.

Editing Parameters of an Existing Action

To edit the configuration parameters of an existing Action:

1. Double-click the left side of the Action as shown below.



The system displays the Edit Action Association dialog box for the Action.

2. Enter the information for configuring the Action as described in "Action" in the "SFC Program Elements" chapter. Then click **OK** to save your changes.

Deleting an Action

You cannot delete an Action from the Action Manager if there are still references to the Action in the SFC.

If you delete the last reference to an Action, the system will prompt you to delete the Action from the Action Manager. If you confirm, the Action is deleted automatically from the Action Manager.

To delete an existing Action permanently:

1. Select the Action in the SFC and click the **Cut tool** to delete the Action.
2. On the **Tools** menu, click **Action Manager.** The **Action Manager** dialog box appears.
3. Select the Action to delete and click the **Delete** button. The Action is removed from the Action Manager.

Note You must delete the Action from both the Action Manager and the SFC to remove the Action code completely from your program. However, it is not necessary to delete the Action if you do not want it to execute.

Renaming an Action

If you rename an Action from the Action Manager, all references to that Action are updated to reflect the new name.

To rename an existing Action:

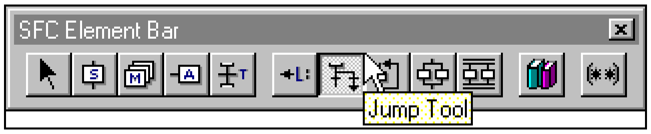
1. On the **Tools** menu, click **Action Manager.** The **Action Manager** dialog box appears.
2. Select the Action to rename and click the **Rename** button.
3. Enter the new name.

Adding a Jump

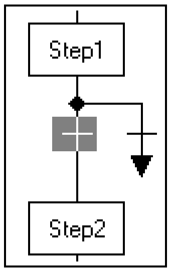
A Jump-to-Label combination is available that allows SFC execution to transfer to any location indicated by a Label element. See "Jump/Label: Program Flow" in the "SFC Program Elements" chapter for detailed information about using Jumps and Labels.

To add a Jump to the program:

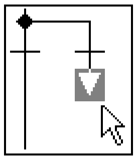
1. Select or deselect **Boolean Transition** on the **Edit** menu to choose the type of Transitions to use with the Jump: RLL or Boolean.
2. Click **Jump Tool** on the SFC tool bar. The cursor becomes the Jump Tool cursor.



3. Move the cursor to the location in the program where you want to place the new Jump and click. The new Jump and two Transitions appear in the program.

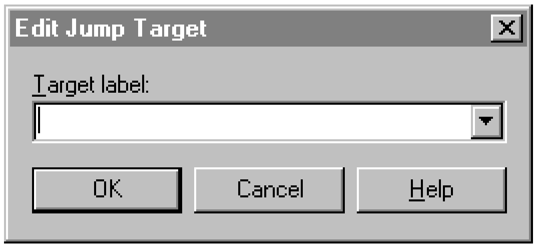


4. Double-click the arrowhead of the Jump.



Editing the Jump Target

The **Edit Jump Target** dialog box appears.



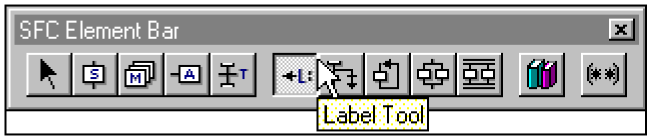
5. Enter the label to which the Jump transfers program flow. Labels must start with an alphabetical character and be followed by any alphanumeric characters and/or underscore. Labels are not case-sensitive.
6. Edit the two Transitions as described in "Adding a Transition."

Adding a Label

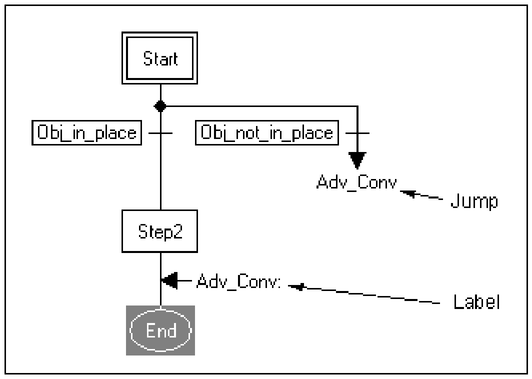
A Jump-to-Label combination is available that allows SFC execution to transfer to any location indicated by a Label element. See "Jump/Label: Program Flow" in the "SFC Program Elements" chapter for detailed information about using Jumps and Labels.

To add a Label to the program:

1. Click the **Label Tool** on the SFC toolbar. The cursor changes into the Label Tool cursor.

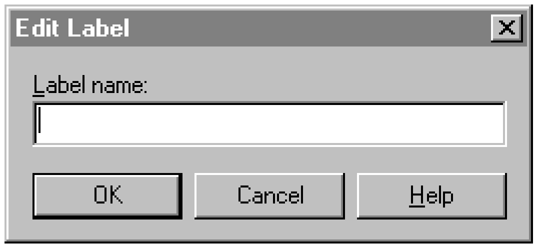


2. Move the cursor to the location in the program where you want to place the new Label and click. The new Label appears in the program.



Adding a Label

3. Double-click the Label. The **Edit Label** dialog box appears.



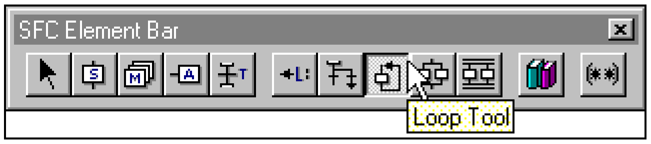
4. Enter a meaningful label. Labels must start with an alphabetical character and be followed by any alphanumeric characters and/or underscore. Labels are not case-sensitive.

Adding a Loop

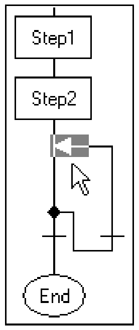
A loop allows the SFC program execution to go back to a preceding location in the program in order to repeat a series of Steps. See "Loop: Program Flow" in the "SFC Program Elements" chapter for detailed information about using Loops.

To add a loop to the program:

1. Select or deselect **Boolean Transition** on the **Edit** menu to choose the type of Transitions to use with the loop: RLL or Boolean.
2. Click **Loop Tool** on the SFC toolbar. The cursor becomes the Loop Tool cursor.

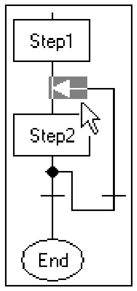


3. Move the cursor to the location in the program where you want to place the lower end of the loop and click. The loop appears in the program.



Adding a Loop 1

4. Drag the loop arrow to the point where the upper end of the loop is to be located.



Adding a Loop 2

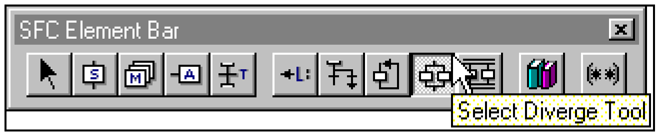
5. Edit the two Transitions as described in "Adding a Transition."

Adding a Select Divergence

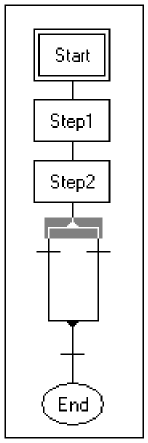
A Select Divergence allows the SFC program execution to follow one of two or more control paths. See "Select Divergence: Program Flow" in the "SFC Program Elements" chapter for detailed information about using Divergences.

To add a Select Divergence to the program:

1. Select or deselect **Boolean Transition** on the **Edit** menu to choose the type of Transitions to use with the divergence: RLL or Boolean.
2. Click the **Select Diverge Tool** on the SFC toolbar. The cursor changes into the Select Divergence Tool cursor.



3. Move the cursor to the location in the program where you want to place the Select Divergence and click. The Select Divergence appears in the program.

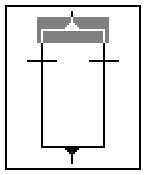


Adding a Select Divergence

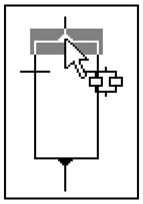
4. Edit the two Transitions as described in "Adding a Transition."

To add another path to the Select Divergence:

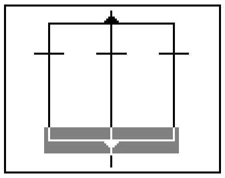
1. Click the top of the Select Divergence.



2. Click **Select Diverge Tool** and place the cursor at the top of the divergence.



3. Click and another divergence path appears.



To delete one path in a divergence:

- Click the path and click the **Cut Tool.**

To delete the entire divergence:

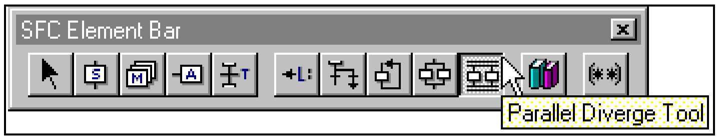
1. Click either the top or the bottom of the divergence.
2. Click the Cut Tool.

Adding a Parallel Divergence

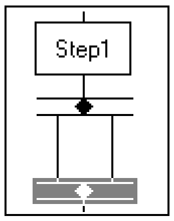
A Parallel Divergence allows the SFC program execution to follow two or more control paths. Execution along each path must be completed for program execution to proceed beyond the Parallel Divergence. See "Parallel Divergence: Program Flow" in the "SFC Program Elements" chapter for detailed information about using Parallel Divergences.

To add a Parallel Divergence to the program:

1. Click **Parallel Divergence Tool** on the SFC toolbar. The cursor becomes the Parallel Divergence Tool cursor.



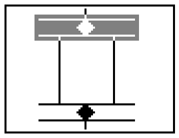
2. Move the cursor to the location in the program where you want to place the Parallel Divergence and click. The Parallel Divergence appears in the program.



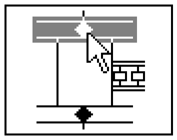
Adding a Parallel Divergence

To add another path to the Parallel Divergence:

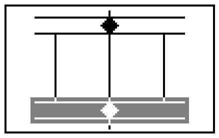
1. Click the top of the Parallel Divergence.



2. Click **Parallel Diverge Tool** and place the cursor at the top of the divergence.



3. Click and another divergence path appears.



To delete one path in a divergence:

- Click the path and click the **Cut Tool.**

To delete the entire divergence:

1. Click either the top or the bottom of the divergence.
2. Click the **Cut Tool.**

Adding a Library Step

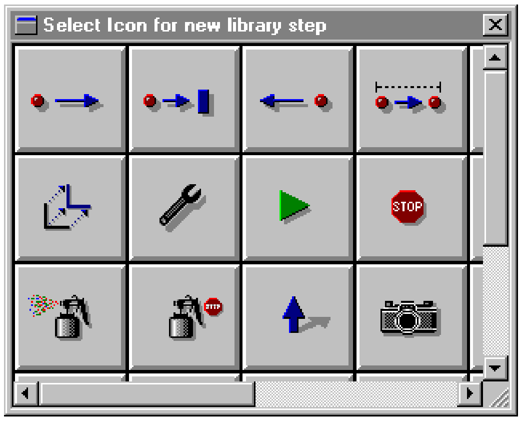
A library Step is a program Step containing a code template, which is user-defined for a specific function, and an icon, which is appropriate for the function. The Step itself operates like any other Step, based on the program code that you use in it. InControl provides a library of several icons that can be used within an SFC. See "Step" in the "SFC Program Elements" chapter for detailed information about using Steps in the program.

You must create a Step for the library before adding it to a program.

Building the Step Library

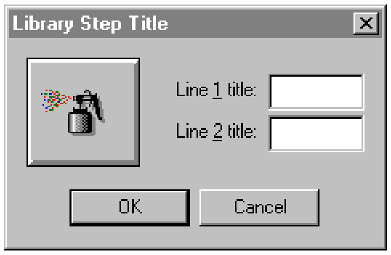
To create a new Library Step for the library:

1. On the **Tools** menu, click **Step Library,** and then click **New Library Step.** The palette of icons appears.

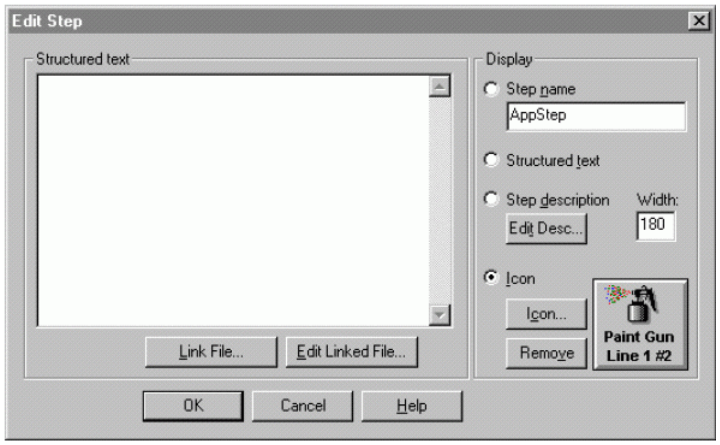


Step Library Palette of Icons

2. Click an icon to represent the Step. The **Library Step Title** dialog box appears.



3. Enter a title for the Step and click OK. The **Edit Step** dialog box appears.



Edit Step Dialog Box

4. Customize the Step:

   Enter the Structured Text code.

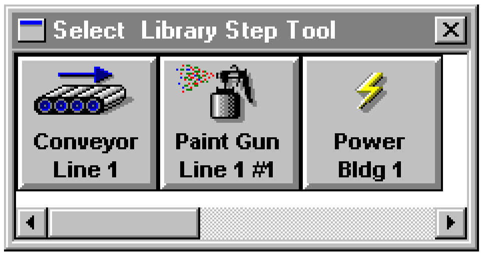
   Enter Step name

   Choose how you want to display the Step: icon, Step name, Code, etc.

5. Click **OK.** The new Step is added to the library, and the palette of icons appears. Either create a new Step Template or close the palette.

To edit a Step in the library:

1. On the **Tools** menu, click **Step Library,** and then click **Edit Library Step.** The **Edit Library Step** window appears.



2. Click the Step to edit.

3. After the **Edit Step** dialog box appears, make your changes and then click **OK.** The changes are added to the Step Template.

4. The **Edit Library Step** palette of icons appears. Either edit another Step Template or close the palette.

To delete a Step Template from the library:

1. On the **Tools** menu, click **Step Library,** and then click **Delete Library Step.** The **Delete Library Step** window appears.

2. Click the Step to delete. Note that you are not prompted to confirm.

3. The selected Step is removed from the library.

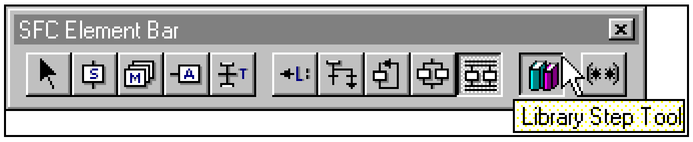
4. The **Delete Library Step** window appears. Either delete another Step or close the window.

Adding a Step from the Library

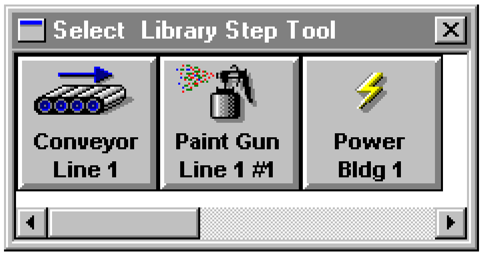
You must create a Step for the library before adding it to a program

To add a Step from the library to a program:

1. Click **Library Step** on the SFC toolbar.



The **Select Library Step Tool** window appears.



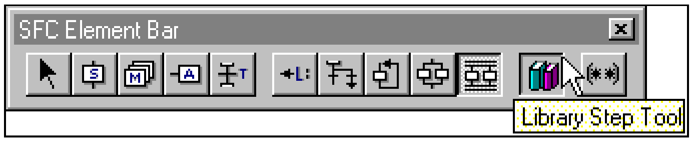
2. Click the appropriate Step. The menu closes and the cursor changes to the Library Step cursor.
3. Move the cursor to the location in the program where you want to place the Step and click once. The **Edit Step** dialog box appears.
4. Enter the appropriate information for displaying the Step in the SFC and click **OK.**

Bitmap Library Editor

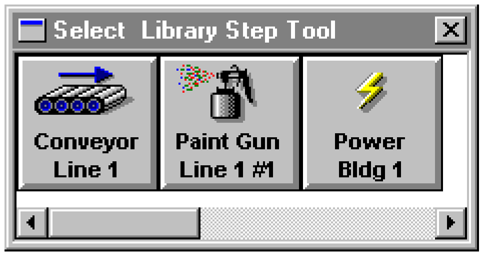
The Bitmap Library Editor allows you to add bitmaps to the SFC Step or SFC Macro step.

To add bitmaps to the SFC Step or SFC Macro Step:

1. Click **Library Step** on the SFC toolbar.

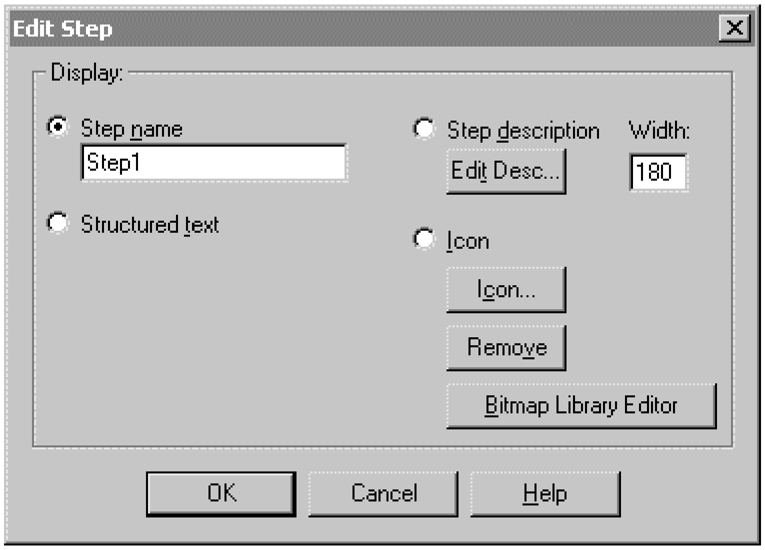


The **Select Library Step Tool** window appears.



2. Click the appropriate step. The menu closes and the cursor changes to the Library Step cursor.

3. Move the cursor to the location in the program where you want to place the Step and click once. The **Edit Step** dialog box appears.


Edit Step

Display:

(● Step name) Step1

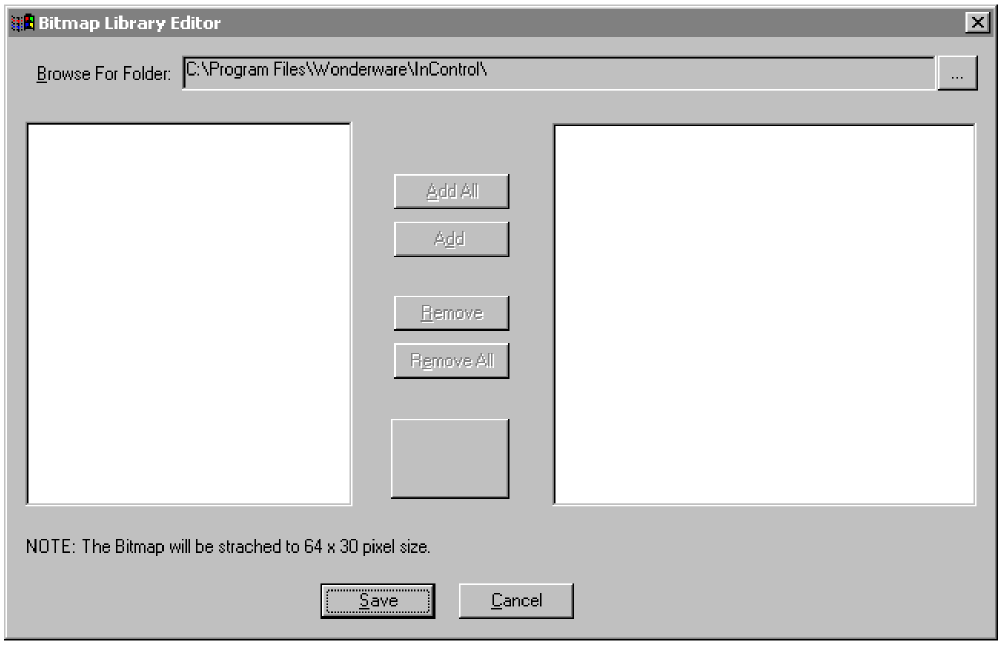
○ Structured text

○ Step description — Edit Desc... — Width: 180

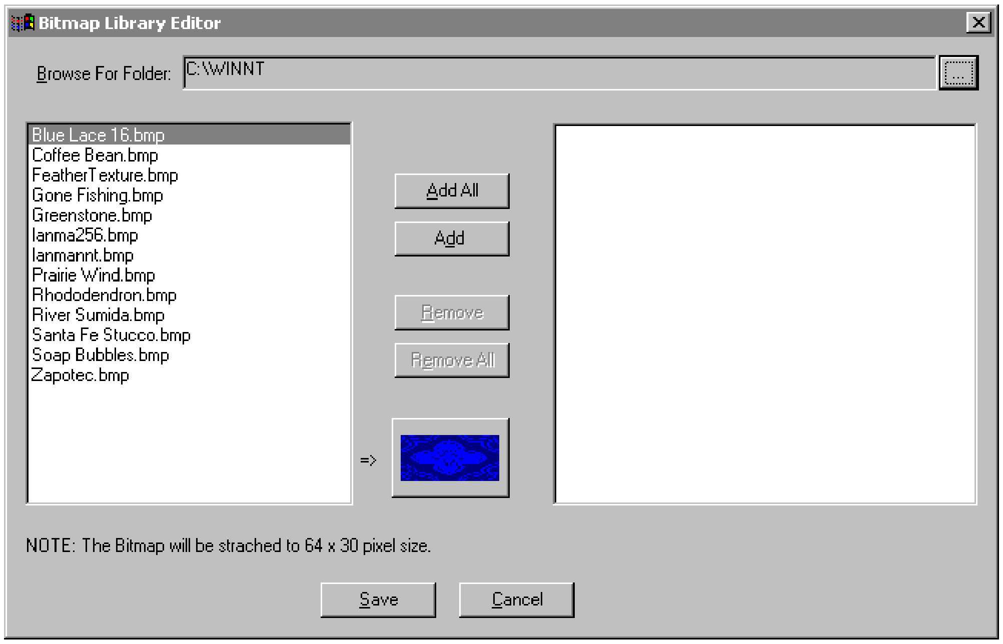
○ Icon — Icon... — Remove — Bitmap Library Editor

OK — Cancel — Help

4. Click **Bitmap Library Editor**. The **Bitmap Library Editor** dialog box appears.

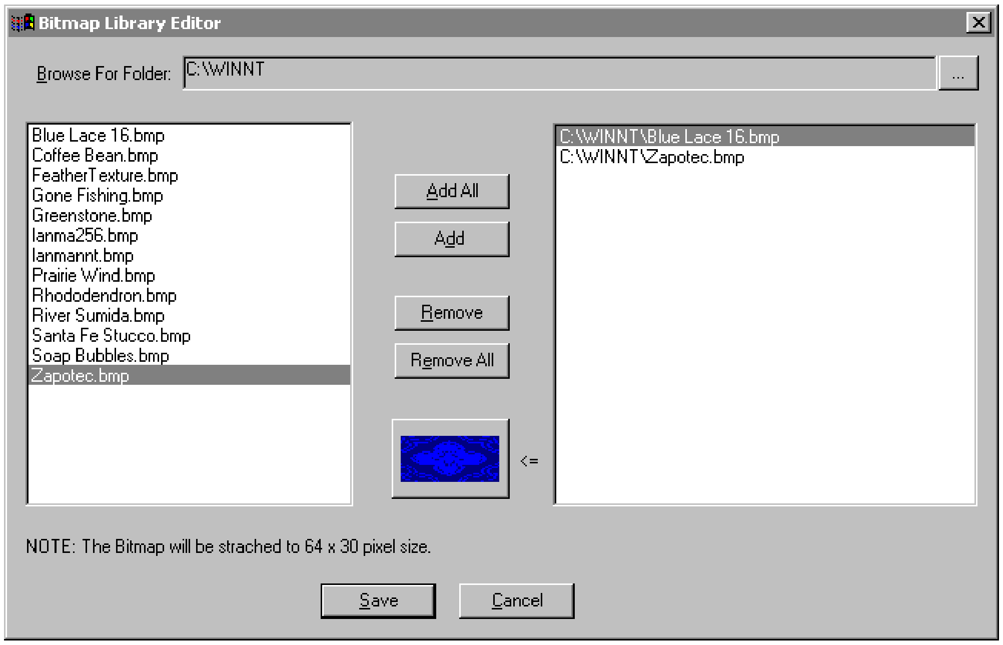

Bitmap Library Editor
Browse For Folder:
C:\Program Files\Wonderware\InControl\
...
Add All
Add
Remove
Remove All
NOTE: The Bitmap will be strached to 64 x 30 pixel size.
Save
Cancel

5. Click the ellipses to browse for bitmaps. The available bitmaps will be listed in the left list box.



Blue Lace 16.bmp
Coffee Bean.bmp
FeatherTexture.bmp
Gone Fishing.bmp
Greenstone.bmp
lanma256.bmp
lanmannt.bmp
Prairie Wind.bmp
Rhododendron.bmp
River Sumida.bmp
Santa Fe Stucco.bmp
Soap Bubbles.bmp
Zapotec.bmp

6. Click the desired bitmaps, then click **Add** or click **Add All** to add bitmaps in Library Editor.



Browse For Folder: C:\WINNT

Blue Lace 16.bmp
Coffee Bean.bmp
FeatherTexture.bmp
Gone Fishing.bmp
Greenstone.bmp
lanma256.bmp
lanmannt.bmp
Prairie Wind.bmp
Rhododendron.bmp
River Sumida.bmp
Santa Fe Stucco.bmp
Soap Bubbles.bmp
Zapotec.bmp

C:\WINNT\Blue Lace 16.bmp
C:\WINNT\Zapotec.bmp

NOTE: The Bitmap will be strached to 64 x 30 pixel size.

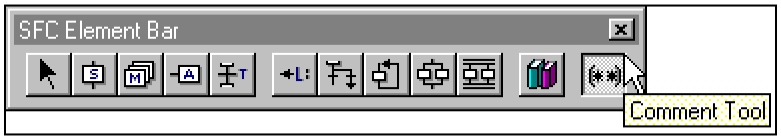
7. Click **Save**.
8. Click the **Icon** radio button. The bitmap added in Bitmap Library Editor is added in the Attach New Icon window.
9. Select the bitmap or icon to display in the Step or Macro Step.

Adding Program Comments

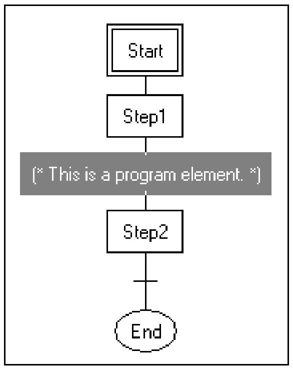
A program comment can consist of any meaningful description that you want to display adjacent to a program element. You can choose whether the system displays the comments or hides them.

To add a comment to the program:

1. Click the **Comment Tool** on the SFC toolbar. The cursor changes into the Program Comment Tool cursor.



2. Move the cursor to the location in the program where you want to place the comment and click. The new Comment element appears in the program.
3. Double-click the Comment element. The **Program Comments** dialog box appears.
4. Enter the comment and click **OK.** The comment appears in the program.

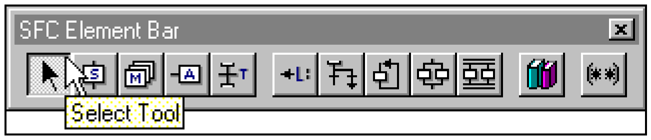


SFC Comment

Editing Program Elements

To edit an existing program element:

1. Click the Select Tool.



2. Double-click the element (Step, Transition, label, etc.). The dialog box appropriate for the element **(Edit Step, Select RLL Transition Logic, Bypass Jump Transition Logic,** etc.) appears.

   If an element is already selected (highlighted), you can also open the dialog box for editing by pressing **Enter.**

3. Make changes in the dialog box as needed.

C H A P T E R 3

SFC Program Elements

This chapter introduces the programming elements that you use in a Sequential Function Chart.

Contents

Elements of the SFC

The Sequential Function Chart (SFC) represents an application program as a series of sequential states or Steps. A Step represents a condition in which the program execution follows a set of rules defined by the Actions and functions associated with the Step.

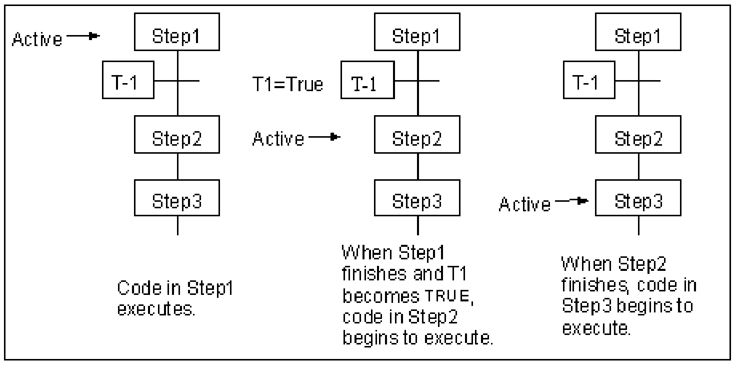
You program a Step in the Structured Text language (IEC-61131 compliant) described in detail in the "Structured Text Language" chapter.

The flexibility of the SFC language syntax allows you to call another entire SFC (the child SFC) for execution from within a single Step, termed the Macro Step. When the child SFC has completed, program control returns to the Macro Step that made the call.

You can design multiple branches in your SFC. The Select Divergence branch can consist of two or more paths, and program execution is allowed to follow only one of the paths. The Parallel Divergence branch also consists of multiple paths, but program execution proceeds down all the paths.

Program Flow

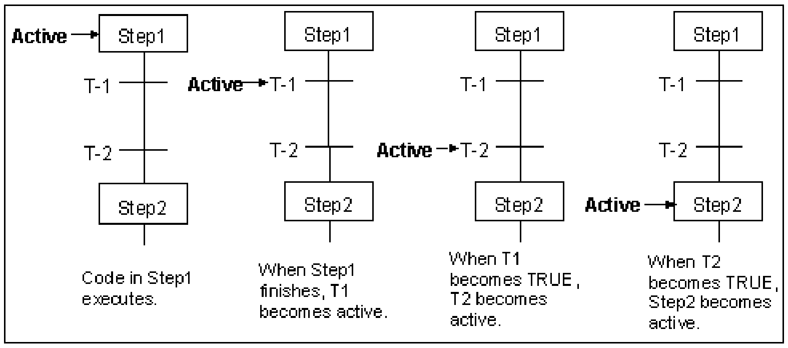
Program flow moves from top to bottom, as illustrated in the following figure. The code within each Step is executed, and when it has completed, program flow moves to the next program element. If the next element is a Step, the code within that Step is executed. If the next element is a Transition, program flow continues when the Transition becomes TRUE. A Transition is a Boolean language or RLL language condition that resolves to a TRUE or FALSE state.


Active
Step1
T-1
Step2
Step3
Code in Step1 executes.
Step1
T1=True
T-1
Active
Step2
Step3
When Step1 finishes and T1 becomes TRUE, code in Step2 begins to execute.
Step1
T-1
Step2
Active
Step3
When Step2 finishes, code in Step3 begins to execute.

Program Flow Example 1

When program flow reaches the End Step, the mode of the SFC changes from Run to Complete. The program must be restarted before it can execute again.

In the following example, one Transition follows another. Program flow still moves from top to bottom, and execution of a program element does not begin until the preceding element has completed. After the first Transition, T1, becomes TRUE, the second Transition T2 becomes active. After the second Transition, T2, becomes TRUE, Step2 becomes active.


Active
Step1
T-1
T-2
Step2
Code in Step1 executes.
Active
Step1
T-1
T-2
Step2
When Step1 finishes, T1 becomes active.
Active
Step1
T-1
T-2
Step2
When T1 becomes TRUE, T2 becomes active.
Active
Step1
T-1
T-2
Step2
When T2 becomes TRUE, Step2 becomes active.

Program Flow Example 2

You can use a Jump-to-Label combination to skip code as your application requires it. You can also incorporate a Loop to re-execute a section of code.

InControl allows you to execute multiple programs of multiple types. That is, you can run two RLL programs, for example, at the same time that three SFCs are running. You can coordinate program execution through global symbols, which are recognized by all the program types. The system variable, Mode, which is associated with an individual program, is a local symbol that you can also use to coordinate program execution.

SFC Extensions to IEC 61131-3

This section describes the enhancements and other extensions to the IEC 61131-3 specification. InControl complies with IEC 61131-3 except where noted here.

Transitions

Typically, Steps in an SFC are separated with Transitions. InControl allows you to place a Step immediately before or after another Step. You can also place one Transition after another, with no Steps between them.

When two Steps are not separated by a Transition, InControl inserts an invisible TRUE Transition between them. When two Transitions are not separated by a Step, InControl inserts an invisible empty Step between them.

All functions within all the Steps preceding a Transition must have been completed before the system evaluates a Transition.

Step Representation

Steps can be represented by a box containing the Step name, the code programmed within the Step, or by an icon.

Transition Coil

You can use a Transition Coil within the logic of an Action to stop all associated Steps and transfer control to the associated Label.

Macro Step

You can use the Macro Step to call one SFC for execution from a Step in another SFC. Program flow transfers to the SFC that was called (the child SFC). When the child SFC has completed execution, program flow returns to the parent SFC and resumes after the Macro Step

Since you use STL code in an SFC program, the STL enhancements and extensions to the IEC 61131-3 specification are also listed in this section. InControl complies with IEC 61131-3 except where noted here.

Parameters

The parameters for Structured Text functions can be listed in any order as long as the formal parameter names are given as specified by IEC-61131-3.

FOR Statement

You can use the END_FOR_NOWAIT statement to loop back without an I/O scan.

REPEAT Statement

You can use the END_REPEAT_NOWAIT statement to loop back without an I/O scan.

SCAN Statement

You can use the SCAN statement to suspend the execution of Structured Text statements until after the next I/O scan.

WHILE Statement

You can use the END_WHILE_NOWAIT statement to loop back without an I/O scan.

Statement

You can use the BREAK statement to stop program flow. BREAK is useful for debugging a program.

Unsupported Functions

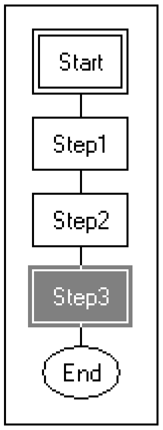
InControl does not support the following functions, which are defined in the

IEC 61131-3 specification.

- LIMIT
- MUX
- SEL

Step

The SFC represents an application program as a series of sequential Steps. A Step represents a condition in which the behavior of the factory process is defined by the Structured Text code and the Actions associated with the Step. While the SFC is being executed, a Step is either active or inactive, and at a given moment, the state of the factory process is defined by the active Steps and the values of their internal and output variables. In the figure below, for example, Step3 is the active Step, and only the code within Step3 is being executed. You can create SFCs with multiple paths, however, and it is possible for more than one SFC to be active at a time.



SFC Step

A Step is represented within an SFC as a box containing the Step identifier. Program flow into and out of the Step is through a vertical line entering the top of the box and another line exiting from the bottom of the box.

When you create a new SFC, the system automatically generates the first Step, labeled Start, and the last Step, labeled End. You cannot edit these Steps; they simply represent the initiation and termination of the SFC.

Typically, you separate Steps in an SFC with Transitions, which are program elements described in "Transition." As an enhancement to the IEC- 61131 specification, InControl allows you to place a Step immediately before or after another Step, which implies a TRUE Transition exists between them.

The Structured Text code in a Step is executed one time, top to bottom, when the Step becomes active. The code is not executed again until the Step becomes inactive and then active again. This occurs in one scan; however, the presence of loops, file operations, or SCAN statements can cause the execution to take more than one scan.

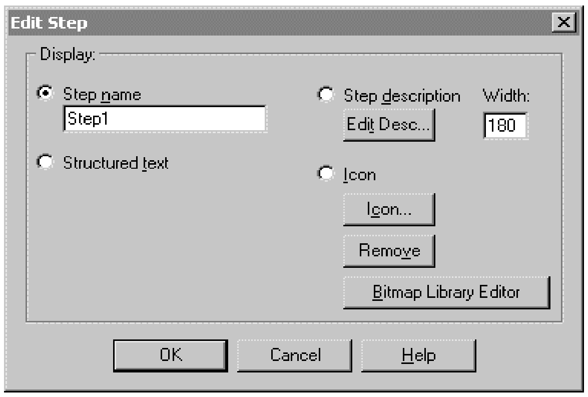
If you want to execute code continually while a Step is active, you must place it in an Action. An Action is another program element that you can use to coordinate the execution of code within a Step with other program code. For more information, see "Action."

Parameters

The Step parameters define how the Step is displayed in the program.

To edit the parameters of a Step:

- On the **Edit** menu, click **Step Properties.** The **Edit Step** dialog box appears. You can also right-click the Step.



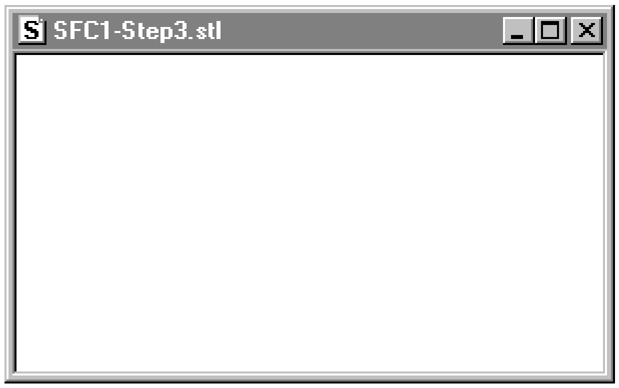
Edit Step Dialog Box

Button / Field	Description
Step Name	Enter a name for the Step. Click the **Step Name** radio button to display the Step name in the SFC.
Structured Text	Click the **Structured Text** radio button to display the Step code in the SFC. For a long series of commands, this can enlarge the displayed size of the Step significantly.
Step Description	Click the **Step Description** radio button to display the Step description in the SFC.
Edit Description	Click **Edit Description** to enter a description for the Step.
Width	Enter the width in pixels for the Step description.
Icon (radio button)	Click the **Icon** radio button to display the Step in the SFC as an icon.
Icon (button)	Click the **Icon** button to display the palette of icons from which to choose. Click the icon and enter a title when the system prompts you.
Remove	Click **Remove** to delete an icon from the Step, if one is assigned.
Bitmap Library Editor	Click **Bitmap Library Editor** to add custom bitmaps to the palette of icons.

Code

To enter code in a Step:

1. Double-click the Step. A Structured Text editor window opens.



2. Enter the program code using the Structured Text language.

When you enter code used in SFC Steps, you use the same tools and menu options that are in the STL editor.

Using Library Steps

A library Step is a program Step containing a code template, which is user-defined for a specific function, and an icon, which is appropriate for the function. The Step itself operates like any other Step, based on the program code that you use in it. InControl provides a library of several icons that can be used within an SFC.

Access the library by clicking **Library Step** on the SFC toolbar. After placing a library Step into an SFC, edit it in the same way as you edit a normal Step. Since the code is a template, you need to modify the code to suit your application. You can also make other changes to the Step options, choosing from the options described in the preceding pages.

For more information about building the Step Library, see "Building the Step Library" in the "Using the SFC Editor" chapter.

Using the SFC and Step System Variables

For each SFC, InControl creates the DN system variable, which you can use to help coordinate program execution. The SFC DN variable is TRUE when the SFC is finished executing. To reference the variable, enter the SFC name followed by a period and the variable suffix. For example, SFC1.DN refers to SFC1.

For each SFC Step, InControl creates three system variables:

- The Boolean Step-is-active (X) variable is TRUE when the Step is active and FALSE when the Step is inactive.
- The Step code is done (DN) variable is TRUE when the code inside the Step has completed execution.

- The Step time (T) variable contains the current elapsed time of the Step in milliseconds. When a Step is inactive, T contains the total elapsed time of the Step. T is set to zero when the Step becomes active.

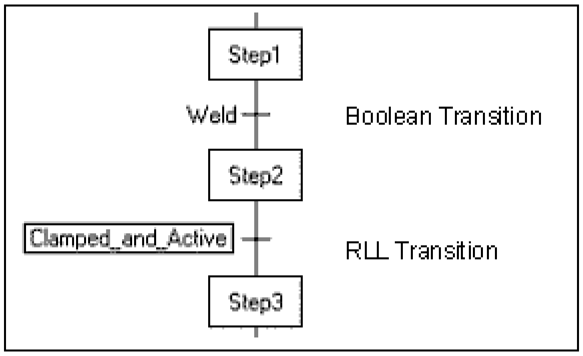
You can use these system variables in any expression, contact or coil instead of a symbol of the same type. To reference a Step variable within the program, enter the Step name followed by a period and the variable suffix. For example, STEP1.X refers to the Step active variable for Step STEP1.

You can add the Step variables to the Watch Window. Use the following naming format to add a variable to the Watch Window:

```
<programname>.<Stepname>.<systemvariablename>
```

Transition

A Transition represents the condition that allows program flow to pass from one or more Steps preceding the Transition to one or more Steps following the Transition. When the system evaluates the code comprising a Transition, the result must be either TRUE or FALSE. In the figure below, for example, program flow has passed the Boolean Transition and Step2, which follows it, and is currently at the RLL Transition. Until the RLL Transition evaluates to TRUE, Step3 cannot execute.



SFC Transition

A Transition is represented, as shown in the preceding figure, as either a horizontal line with the Boolean code adjacent to it (Boolean Transitions), or a horizontal line with the name of the RLL output coil contained within a box (RLL Transitions). Program flow in and out of the Transition is through a vertical line passing through the horizontal line.

You can define a Transition by either of the following methods:

- A Boolean Transition is a Boolean expression composed of Structured Text.
- An RLL Transition consists of a single RLL rung with an output coil having the same name as the Transition itself.

Typically, all Steps are separated by Transitions. As an enhancement to the IEC-61131 specification, InControl allows you to place a Step immediately before or after another Step with no Transitions to separate them, or multiple Transitions between two Steps. No Transition between Steps implies a TRUE transition; no Step between Transitions implies an empty Step.

Evaluation

When program flow in an SFC encounters a Transition, evaluation occurs as follows.

- RLL Transition When power flow on the rung reaches the output coil, turning it on, the Transition becomes TRUE, and program flow moves to the next Step.

  Boolean Transition When the Boolean expression resolves to TRUE, the Transition becomes TRUE, allowing program flow to move to the next Step.
- All functions within all the preceding Steps must have been completed before the system evaluates a Transition. This is a further enhancement to the IEC61131 specification, which only requires all preceding Steps to be active before a Transition can be evaluated.
- If a Transition is FALSE and remains FALSE, the system does not re-execute the Structured Text code in the Steps that precede the Transition. Program flow remains at the Transition until the Transition becomes TRUE.

Parameters

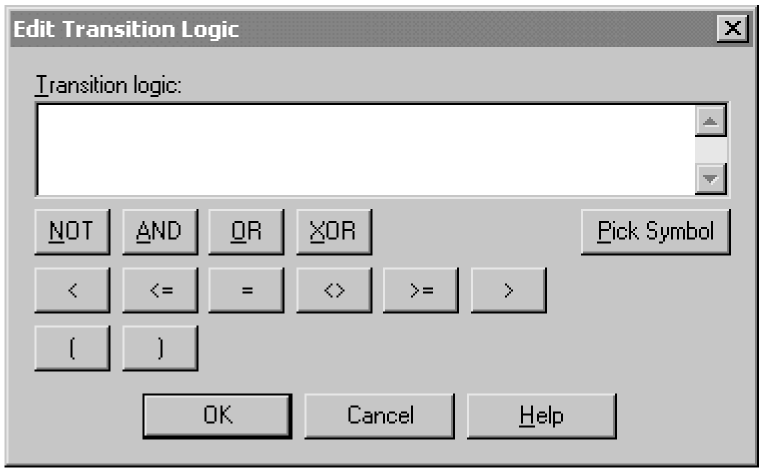
To edit a Transition, double click the Transition. Enter your code as described below.

RLL Transition For a new Transition, a dialog box appears in which you enter the Transition name. The system automatically assigns this name to the variable name of the output coil and opens the RLL editor. An RLL Transition can have the same name as an Action. However, the RLL logic for Transitions and for Actions is scoped differently. Therefore, using the same name for a Transition does not mean that the same RLL logic is executed for the Action, and vice versa.

Enter the RLL logic through the RLL editor, using the same rules for contacts, coils, Jumps, etc., described in the "Relay Ladder Logic Program Elements" chapter. Recall that you can program only one rung for an RLL Transition.

Note You can use any of the InControl predefined functions or function blocks in an RLL Transition, but not use a user-defined function or function block.

Boolean Transition When you edit the **Edit Transition Logic** dialog box, you can either type the Boolean expression directly into the **Transition Logic** field, or click the buttons to select operators and symbols. You can access the Symbol Manager to configure local variables and to see a list of all configured variables that you can use in the Boolean expression.

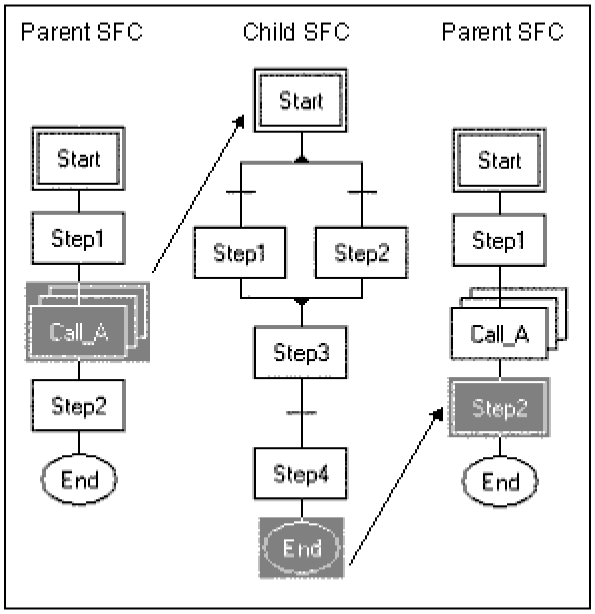


SFC Edit Transition Logic Dialog Box

Macro Step

The Macro Step provides a means of calling one SFC for execution from a Step in another SFC. Program flow transfers to the SFC that was called (the child SFC). When the child SFC has completed execution, program flow returns to the calling (parent) SFC and resumes after the Macro Step.

In the following figure, for example, call_A is the Macro Step in the parent SFC that calls the child SFC for execution. When the child SFC completes execution, program flow resumes at Step_2 in the parent SFC.



SFC Macro

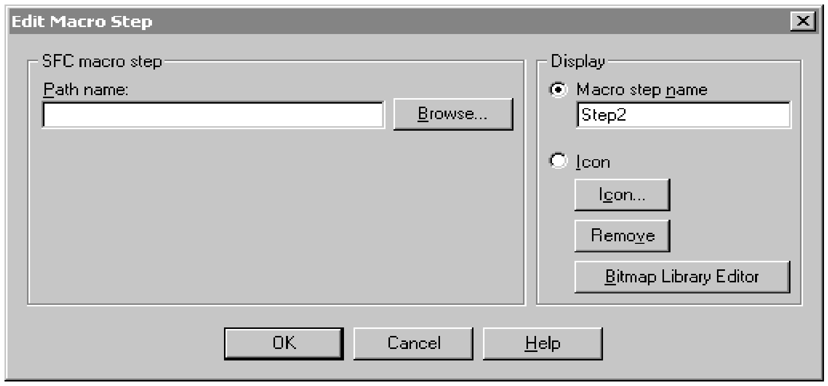
Representation of a Macro Step is similar to a Step, multiple boxes containing an identifier. Program flow into and out of the Macro Step is through a vertical line entering the top, and another line exiting from the bottom.

Parameters

The Macro Step parameters define how the Macro Step is displayed in the program.

To edit the parameters of a Macro Step:

- On the **Edit** menu, click **Step Properties.** The **Edit Macro Step** dialog box appears. You can also right-click the Macro Step.



Edit Macro Step Dialog Box

Button / Field	Description
Path Name	Enter the name of the SFC file containing the program code for the child SFC. If you are unsure of the file name, click **Browse.**
Browse	Click to locate the SFC file that contains the program code for the child SFC.
Macro Step Name	Enter a name for the Macro Step. Click the **Macro Step Name** radio button to display the Macro Step name in the SFC.
Icon (radio button)	Click the **Icon** radio button to display the Macro Step in the SFC as an icon.
Icon (button)	Click the **Icon** button to display the palette of icons from which to choose. Click the icon and enter a title when the system prompts you.
Remove	Click **Remove** to delete an icon from the Macro Step, if one is assigned.
Bitmap Library Editor	Click **Bitmap Library Editor** to add custom bitmaps to the palette of icons.

Code

To specify code for the Macro Step:

1. Double-click the Macro Step. The SFC editor opens and displays the SFC specified in the **Path Name** field of the **Edit Macro Step** dialog box.
2. Enter the child SFC program.

Macro Step Usage Rules

When you design a parent/child SFC combination, follow the rules below.

- A child SFC cannot call the parent SFC or itself.
- You can nest SFCs. That is, one child SFC can call another child SFC. There is no limit to the nesting level.
- You can call a child SFC from multiple points within a program.
- If you open a project developed under InControl 7.0, you have the option of converting the files to an InControl 7.1 project. Any macros in that project appear in the Programs folder after the conversion. You can move these macros to the Macros folder, but this is not required for the project to compile. Macros in the Rel. 7.0 project that have been excluded from download and that are called from another SFC will appear in the Macros folder after the conversion.

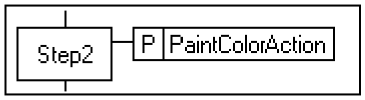
Action

An Action consists of a segment of RLL program code that is associated with a Step or Macro Step. The Action is executed when the Step becomes active, based on the Action qualifier, which determines when the RLL runs relative to the activation of the Step. Note that if a Structured Text Label statement is used within the Step, this can also affect when an Action is executed. If a Label statement is specified, the Action does not run until the Label statement in the Step code is encountered.

When an Action terminates, it is executed one more time on the following scan, with the rung input set to FALSE. This allows timers, counters, and output coils to reset. If you want additional logic to be executed when the Action is terminated, use the F_TRIG function block to negate the FALSE power flow into the rung.

In the example below, the Action called PaintColor consists of several rungs of RLL that are executed when Step2 becomes active. In this particular example, the RLL execution does not begin until code execution in the Step encounters the Label called label_a. The P code is the Action qualifier and means that the RLL is pulsed, that is, it is executed one time only.

For more information about the qualifiers, see "Choosing Action Name."



SFC Action

You can associate zero or more Actions with a Step and you can associate one Action with more than one Step by referencing the Action’s name.

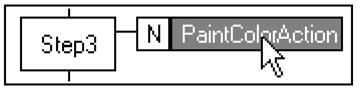
The operation of the SFC transition coil, which can be used in an Action, is described in "SFC Transition Coil" of the "RLL Program Elements" chapter.

The SFC transition coil is useful for implementing emergency shut-down procedures. See "Designing a Safe State."

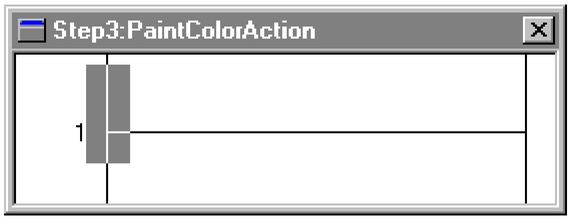
Editing the Action RLL

To edit the RLL code of an Action:

1. Double click the right side of the Action.



The RLL editor window appears.

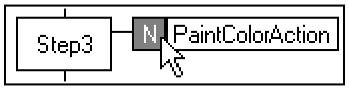


2. Enter the RLL code as described in the "Using the RLL Editor" chapter.

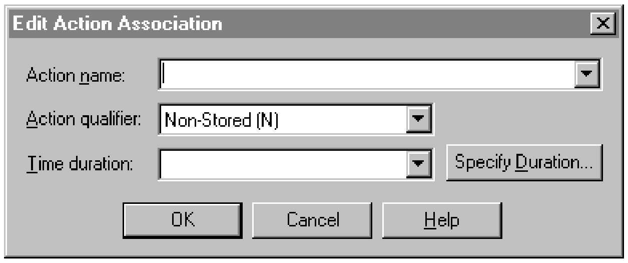
Parameters

To edit the configuration parameters of an Action:

- Double click the left side of the Action.



The **Edit Action Association** dialog box appears.



Edit Action Association

Action name:

Action qualifier: Non-Stored (N)

Time duration:

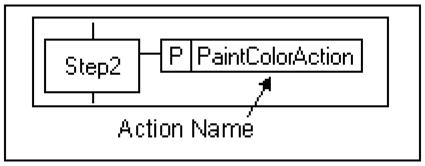
Edit Action Association Dialog Box

Button / Field	Description
Action Name	Enter the name of the Action.
Action Qualifier	Specify an Action qualifier. For more information, see "Choosing Action Name."

Button / Field	**Description**
Time Duration	Specify the time duration for Limited and Delay qualifiers. Enter either a literal value or a variable name. You can either enter the time directly, following the IEC-61131 specification, or click **Specify Duration**to fill in the time in a dialog box. For more information, see "Setting Action Duration." If the Action qualifier does not actually use the time duration, any value entered for this parameter is ignored. If you enter a variable name, the variable is checked only once, when the Action is commanded.
Specify Duration	Click to access the **Define Time Duration** dialog box if you do not want to enter the time directly. For more information, see "Setting Action Duration."
Program Label	Optional. Enter the name of the program label. Labels in a Macro Step SFC cannot be referenced from the parent SFC, and the Labels in the parent SFC cannot be referenced by the macro SFC. If no Label is in the Step with which the Action is associated, then the Program Label parameter is ignored. For more information, see "Choosing the Program Label."

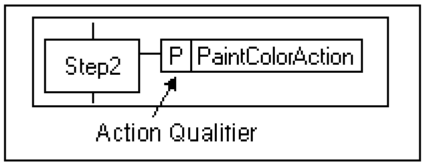
Choosing Action Name

Use this name if you refer to the Action from another Action, such as resetting an Action that was stored in another Action. The Action name appears within the Action as shown in the following figure. An Action can have the same name as an RLL Transition. However, the RLL logic for Actions and for RLL Transitions is scoped differently. Therefore, using the same name for an Action does not mean that the same RLL logic is executed for the RLL Transition, and vice versa.



Choosing Action Qualifier

Action Qualifiers specify constraints on the execution of the RLL code. Qualifiers appear within the Action as shown in the figure below.



Choose from the following qualifiers.

- Non Stored (N) When the Step becomes active, the RLL begins running and stops when the Step becomes inactive.
- Stored (S) When the Step becomes active, the RLL begins to run and continues until reset by the Reset qualifier.
- Reset (R) You can use the Reset qualifier to terminate the RLL that was started with any of the other qualifiers.
- Pulsed (P) When the Step becomes active, the RLL is executed once.
- Time Delayed (D) When the Step becomes active, there is a delay* and then the RLL begins running. The RLL stops when the Step becomes inactive.
- Time Limited (L) When the Step becomes active, the RLL begins running. The RLL stops when the time limit* expires or the Step becomes inactive.
- Delayed and Stored (DS) When the Step becomes active, there is a delay* and then the RLL is stored and begins running. The RLL continues until reset by the Reset qualifier.

  If another Action qualifier resets the RLL during the delay, the reset has no effect because the RLL has not yet been stored.

  If the Step becomes inactive before the delay completes the RLL is never stored and does not run at all.

- Stored and Time Delayed (SD) When the Step becomes active, the RLL is stored. Then, there is a delay* and the RLL begins running. The RLL continues until reset by the Reset qualifier. If an Action is reset during a delay, then the RLL does not execute since it has already been stored.
- Stored and Time Limited (SL) When the Step becomes active the RLL is stored and then begins to run. After the specified time* the RLL stops running. A Reset qualifier is required to reset the RLL. Otherwise, without the reset, the RLL cannot be run again. If the Step becomes inactive, the RLL will continue to run until the duration times out. To restart the Action, you must reset it first.
- Pulse Width (PW) Operates the same as the Stored and Time Limited Qualifier, with this difference: the Action resets automatically after the duration times out; the Reset qualifier is not necessary to restart the Action.

  * Specify a time in the **Time Duration** field of the **Edit Action Association** dialog box.

If a Step in a child SFC contains one of the following types of stored Actions, the Action does not automatically stop when the child SFC reaches the Complete state:

- Stored
- Pulse Width
- Delayed and Stored
- Stored and Time Limited
- Stored and Time Delayed

Consider using a Reset qualifier in an Action of another Step of the child SFC to stop the stored Action before the child SFC reaches the Complete state.

Setting Action Duration

You can enter the duration directly or click **Specify Duration** and enter time intervals in the dialog box. Enter either a literal value or a variable name. If you enter a variable name, the variable is checked only once, when the Action is commanded.

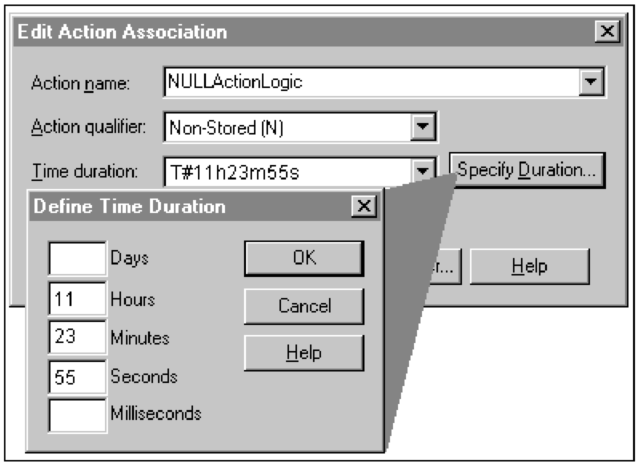
- If you enter the duration directly, follow the IEC 61131-3 specification: a keyword, e.g., T#, TIME#, t#, time#, followed by time in days, hours, minutes, seconds, as shown below.

Action Duration Examples

Time	**Format**	**Time**	**Format**
14.7 days	T#14.7d	4 seconds	Time#4s
2 minutes 5 seconds	T#2m5s	1 day 29 minutes	t#1d29m
14 minutes	time#14m	1 hour 5 seconds 44 milliseconds	T#1h5s44ms

- If you prefer to use the dialog box, enter the time into each field as appropriate.

The figure below illustrates the same time entered by both methods.


Edit Action Association

Action name: NULLActionLogic

Action qualifier: Non-Stored (N)

Time duration: T#11h23m55s

Define Time Duration

Days:

Hours: 11

Minutes: 23

Seconds: 55

Milliseconds:

Setting Duration

Note that if you specify a duration for an Action and choose an Action qualifier that is not time dependent, the duration is ignored.

For an Action that has both a program Label and a duration specified, duration does not begin timing down until after the Step code encounters the program Label.

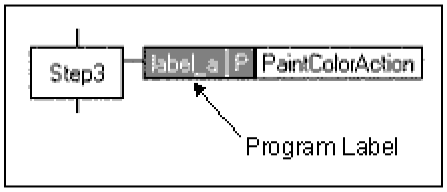
WARNING! If an Action is commanded simultaneously from different Steps, the program may enter the Fault mode if the Action qualifiers are timed; the program may execute unpredictably if other qualifier types are used. This has the potential risk of causing death or injury to personnel and/or damage to equipment. If you use an Action more than once, design the operation of the Action qualifiers so that there is no conflicting execution of the Action code.

Choosing the Program Label

If you specify the optional program Label, the RLL code does not begin running until the code in the Step encounters the Label. Note that in the Structured Text code, the Label must consist of a Label name followed by two colons, as shown below

Label_A::

If you enter a Label, it appears within the Action as shown below.



Designing a Safe State

In the event that an anomaly occurs during program execution, it is useful to divert program flow and have program execution stop, go to a safe state or enter an emergency shut-down procedure, correct an error condition, etc.

A safe-state design consists of two basic sections of code: one section detects the anomalous condition, and the other section responds with special processing as needed. You can create a safe-state design by using one or more Actions to detect the problem condition. Use a transition coil and program Label to jump to the program code that implements the special processing.

Design RLL code within the Action(s) to detect the anomaly. Attach the Action(s) with the detection code to all Steps in which you want to identify the anomalous condition. Using the transition coil allows you to transfer program execution without creating complex branching graphics.

WARNING! Relying exclusively on program code to handle safety-critical emergency conditions has the potential risk of death or injury to personnel and/or damage to equipment. Always install hard-wired mechanical switches, which are independent of solid-state control devices, that can be used for emergency shutdowns.

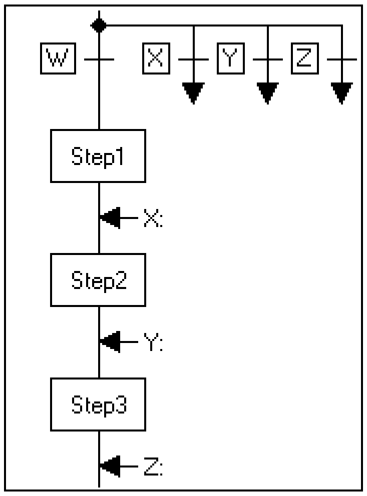
Jump/Label: Program Flow

Program flow can be diverted to a Label from either a Jump program element or an SFC transition coil.

Using a Jump with a Label

The Jump program element allows SFC program flow to transfer to any location indicated by a Label program element. In the following figure, for example, program flow continues to Step1 when condition W is TRUE. When condition X is TRUE and condition W is FALSE, program flow jumps to Label_X, bypassing Step1. When condition Y is TRUE and conditions W and X are FALSE, program flow jumps to Label_Y, bypassing Step1 and Step2. When condition Z is TRUE and conditions W, X, and Y are FALSE, program flow jumps to Label_Z, bypassing Step1, Step2, and Step3.

With multiple branches, logic evaluation takes place from left to right. Program flow follows the first Transition that evaluates to TRUE. If all Transitions are FALSE, program flow halts until one Transition becomes TRUE.



SFC Jump and Label

The Jump is graphically represented by two Transitions: one allows program flow to continue in the downward direction, and the other allows program flow to transfer to a Label identifier, which appears below a directed line to the right and up or down. The Label is graphically represented by a Label identifier and a horizontal line that identifies the point where program flow resumes.

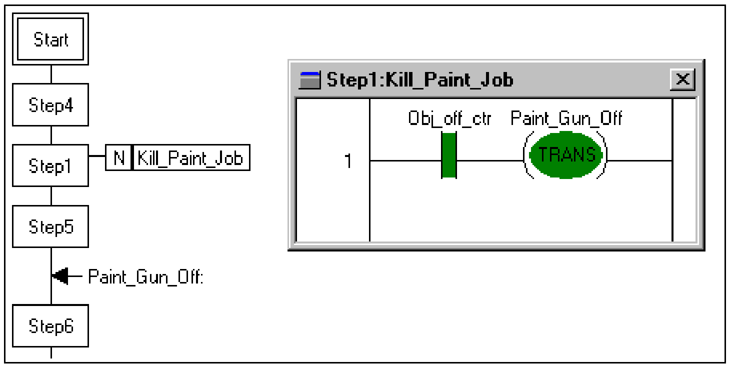
You can define the Transitions for the Jump by either of the following methods.

- A Boolean Transition is a Boolean expression composed of Structured Text.
- An RLL Transition consists of a single RLL rung with an output coil having the same name as the Transition itself.

Using an SFC Transition Coil with a Label

You also divert program flow to a Label from an SFC transition coil used in an Action. When the transition coil receives power flow, program execution transfers to the Label identified within the transition coil.

In the following figure, the SFC transition coil called Paint_Gun_Off has received power flow. The rest of the code in Step 1 is aborted, and program execution resumes at the Label called Paint_Gun_Off, above Step 6.



SFC Transition Coil and Label

For more information about using the SFC transition coil, see "SFC Transition Coil" of the "RLL Program Elements" chapter.

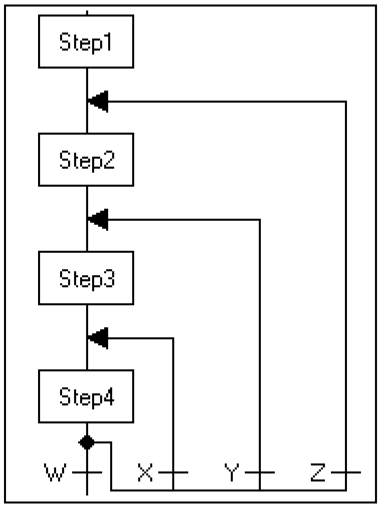
Parameters - Edit Jump and Edit Label Dialog Boxes

Field	Description
Jump: Target Label [1]	Specifies Label to which program flow is transferred.
Label: Label Name [1]	Specifies point in SFC where program flow resumes.
1 Labels must start with an alphabetical character and be followed by any alphanumeric characters and/or underscore. Labels are not case sensitive.	

Loop: Program Flow

In an SFC, program flow usually proceeds from top to bottom. The Loop allows the SFC execution to go back to a previous location in order to repeat a series of Steps. In the figure below, for example, program flow continues to the End when condition W is TRUE. When condition X is TRUE and condition W is FALSE, program flow returns to the point above Step4. When condition Y is TRUE and conditions W and X are FALSE, program flow returns to the point above Step3. When condition Z is TRUE and conditions W, X, and Y are FALSE, program flow returns to the point above Step2.

With multiple branches, logic evaluation takes place from left to right. Program flow follows the first Transition that evaluates to TRUE. If all Transitions are FALSE, program flow halts until one Transition becomes TRUE.



SFC Loop

The Loop is graphically represented by two Transitions: one allows program flow to continue in the downward direction, and the other allows program flow to transfer to a point earlier in the program. An arrow at the top of the Loop shows the point at which program flow resumes.

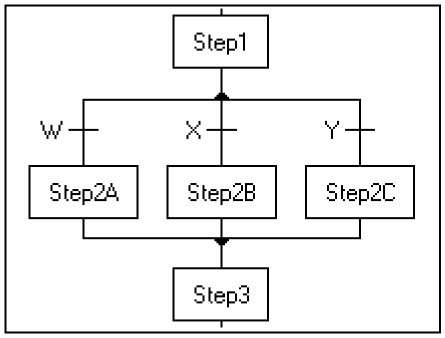
You can define the Transitions for the Loop by either of the following methods.

- A Boolean Transition is a Boolean expression composed of Structured Text.
- An RLL Transition consists of a single RLL rung with an output coil having the same name as the Transition itself.

Select Divergence: Program Flow

You can use the Select Divergence to choose from two or more paths for program flow. Each of the paths within a Select Divergence begins with a Transition condition that determines which path program flow follows. At some point in the SFC all the paths within a Select Divergence must converge. In the following figure, for example, program flow continues to Step2A and then Step3 when condition W is TRUE. When condition X is TRUE, and condition W is FALSE, program flow continues to Step2B and then Step3. When condition Y is TRUE, and conditions W and X are FALSE, program flow continues to Step2C and then Step3.

With multiple paths, logic evaluation takes place from left to right. Program flow follows the first Transition that evaluates to TRUE. If all Transitions are FALSE, program flow halts until one Transition becomes TRUE.



SFC Select Divergence

The Select Divergence is represented as a single path that splits at a horizontal single line into two or more paths with a Transition on each path. The convergence is graphically represented as two or more paths that connect at a horizontal single line.

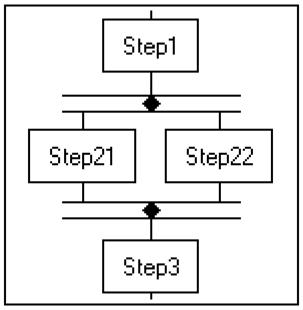
You can define a Transition by either of the following methods.

- A Boolean Transition is a Boolean expression composed of Structured Text.
- An RLL Transition consists of a single RLL rung with an output coil having the same name as the Transition itself.

Parallel Divergence: Program Flow

You can use the Parallel Divergence to allow multiple control paths to be executed simultaneously in parallel. This allows you to design Steps that execute at the same time as other Steps. The Parallel Divergence contains multiple control paths that are all activated as soon as program flow encounters the Parallel Divergence. At some point in the SFC all the paths within a Parallel Divergence must converge. Program flow at the convergence must wait until all the paths have been executed and all paths have arrived at the point of simultaneous convergence.

In the figure below, for example, program flow continues to both Step20 and Step90 simultaneously. When both these Steps have finished execution, program flow continues to Step2.



SFC Parallel Divergence

The Parallel Divergence is graphically represented as a single path that splits at a horizontal double line into two or more paths. The convergence is graphically represented as two or more paths that connect at a horizontal double line.

Rules for Creating Parallel Divergences

Observe the following rules when you create a Parallel Divergence.

- Do not reference the same variable in different paths of a Parallel Divergence.
- Do not call the same child SFC from Macro Steps in different paths of a Parallel Divergence.
- To ensure proper convergence, do not use Labels in the following ways:

  - To jump outside a Parallel Divergence.

  - To jump into a Parallel Divergence.

  - To jump to another path within a Parallel Divergence.

WARNING! Improperly constructed jumps used with Parallel Divergences can cause system lockup, with the potential risk of injury or death to personnel and/or damage to equipment. Be sure to follow these rules when creating a Parallel Divergence.

- Design your code carefully if you modify the same variable in different paths of a Parallel Divergence.

C H A P T E R 4

Structured Text Program Elements

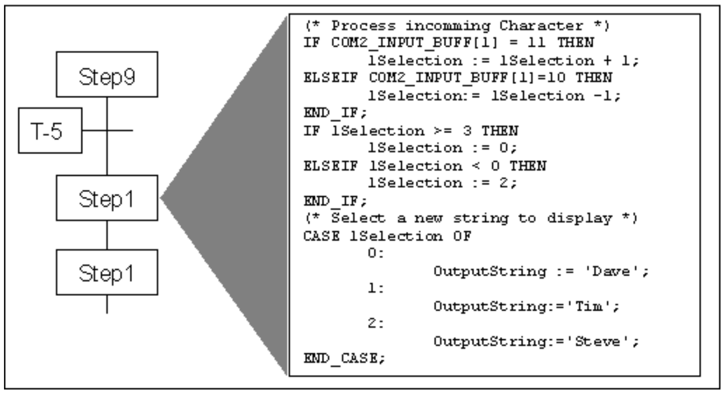
This chapter introduces the Structured Text Language (STL) editor and how to use it to create a new STL program.

Contents

Elements of Structured Text

The Structured Text programming language is a subset of an IEC-61131 compliant set of text-based instructions. These instructions are designed for the easy creation of mathematical and logical operations.

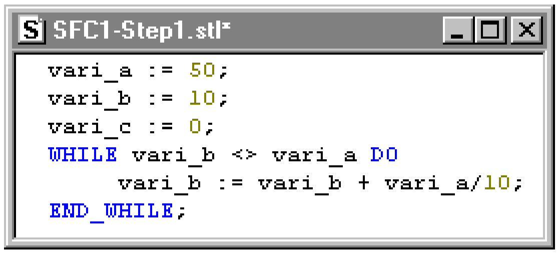
- You use Structured Text when you create the application code for an SFC step, as illustrated below. When the SFC is executed, the Structured Text code that you incorporate within each step is processed as the step becomes active.



```
(* Process incomming Character *)
IF COM2_INPUT_BUFF[1] = 11 THEN
       lSelection := lSelection + 1;
ELSEIF COM2_INPUT_BUFF[1]=10 THEN
       lSelection:= lSelection -1;
END_IF;
IF lSelection >= 3 THEN
       lSelection := 0;
ELSEIF lSelection < 0 THEN
       lSelection := 2;
END_IF;
(* Select a new string to display *)
CASE lSelection OF
       0:
              OutputString := 'Dave';
       1:
              OutputString:='Tim';
       2:
              OutputString:='Steve';
END_CASE;
```

SFC Step Structured Text

- You can create a stand-alone Structured Text program, as illustrated below.



```
vari_a := 50;
vari_b := 10;
vari_c := 0;
WHILE vari_b <> vari_a DO
      vari_b := vari_b + vari_a/10;
END_WHILE;
```

Standalone STL Program

- Currently, any user-defined functions, procedures, or function blocks must be written in the Structured Text language.

As you design the program, keep these points in mind:

- The body of the Structured Text program code itself consists of the "Expressions", "Statement Types", and "Function/Procedure Call" described in this chapter. Valid Structured Text operators and data types are described in "Expressions."
- Make any declarations that you need in the program from the Symbol Manager.
- InControl reserved words have special meaning in the Structured Text Language. Do not use reserved words as variable names.

STL Extensions to IEC 61131-3

This section describes the enhancements and other extensions to the IEC 61131-3 specification. InControl complies with IEC 61131-3 except where noted here.

Parameters

The parameters for Structured Text functions can be listed in any order as long as the formal parameter names are given as specified by IEC-61131-3.

FOR Statement

You can use the END_FOR_NOWAIT statement to loop back without an I/O scan.

REPEAT Statement

You can use the END_REPEAT_NOWAIT statement to loop back without an I/O scan.

SCAN Statement

You can use the SCAN statement to suspend the execution of Structured Text statements until after the next I/O scan.

WHILE Statement

You can use the END_WHILE_NOWAIT statement to loop back without an I/O scan.

BREAK Statement

You can use the BREAK statement to stop program flow. BREAK is useful for debugging a program.

Unsupported Functions

InControl does not support the following functions, which are defined in the IEC 61131-3 specification: LIMIT, MUX, SEL

Unsupported Function Blocks

InControl does not support the following function blocks, which are defined in the IEC 61131-3 specification: SR, RS, SEMA, EDGE_CHECK, RTC.

Additional Built-In Functions and Function Blocks

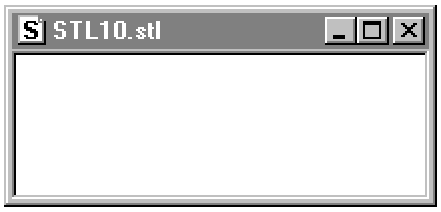
InControl provides the following functions and function blocks, which are not defined in the IEC 61131-3 specification: ARRAY_TO_STRING, STRING_TO_ARRAY, CLOSEFILE, COPYFILE, DELETEFILE, NEWFILE, OPENFILE, READFILE, REWINDFILE, WRITEFILE, MSGWND, ABORT_ALL.

Creating an STL Program

After starting InControl, you can create a new Structured Text program or edit an existing one.

To create a new Structured Text program:

1. On the **File** menu, click **New.**
2. The menu of program types supported by InControl appears.
3. Select **Structured Text,** choose a program type (Program, Function Block, Function) and click **OK.** The **Save As** dialog box appears.
4. Choose a name (up to 31 characters) and directory (project) for the program and click **Save.** A Structured Text edit window appears.



5. To enter the program code, see "Entering Program Code."

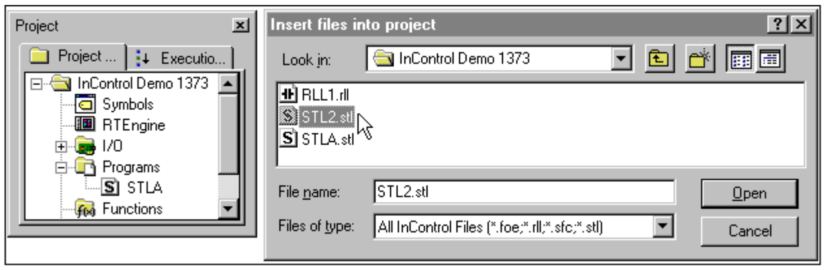
To edit an existing Structured Text program:

1. If the Project window is not open, click **Project** in the **View** menu. The Project window appears.
2. Double-click the name of the program to edit.

   The Structured Text editor opens, displaying the selected program.

You can also click **Open** in the **File** menu to open an existing program for edit. When the **Open** dialog box appears, select the program to open. If a program is not part of the current project, you can add it.

You can click Files into Project in the Insert menu to add any POU (program, function, function block, etc.) to a project. In the following figure, the program STL1, shown in the Insert Files into Project dialog box, is selected and can be added to Project10. Note that the file itself is not copied or moved when it is added to another project.



File name: STL2.stl

Files of type: All InControl Files (*.foe;*.rll;*.sfc;*.stl)

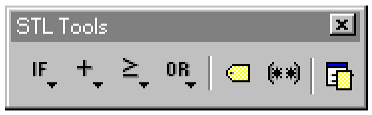
Adding a POU to a Project

Note All POUs are inserted under the Programs folder of the Project window. You must move functions to the Functions folder, function block types to the Function Block folder, and macros to the Macros folder for the project to compile correctly.

If you open a project developed under InControl 7.0, you have the option of converting the files to an InControl 7.1 project. Any macros in that project appear in the Programs folder after the conversion.You can move these macros to the Macros folder, but this is not required for the project to compile. Macros in the Rel. 7.0 project that have been excluded from download and that are called from another SFC will appear in the Macros folder after the conversion.

Using the Structured Text Tool and Menu Bars

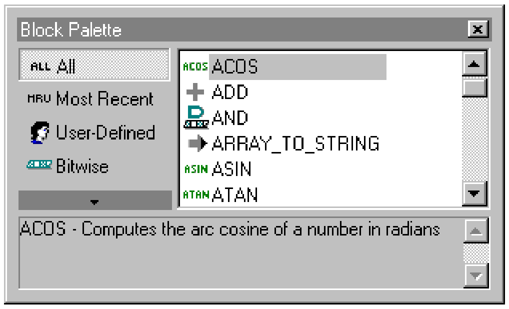
The Structured Text toolbar displays the tools used to create a Structured Text program. The toolbar tools provide shortcuts for entering program elements. You can also type in Structured Text code directly.



Icon	Function	Tool Option
IF	Click to insert preformatted statement types.	If, If Else, If ElseIf Else, Else, ElseIf, EndIf For, For (no wait) Repeat Until, Repeat Until (no wait) While Do, While Do (no wait) Case, Case Item Assignment Exit Include Scan Break
+	Click to insert math operators.	Add + Subtract, Negate - Multiply * Divide / Modulus MODExponentiation ** Assignment : =
≥	Click to insert comparison operators.	Less than < Less than or equal to ~ Equal = Not equal < > Greater than or equal to ~ Greater than >

Icon	Function	Tool Option
OR	Click to insert Boolean operators.	Boolean bitwise AND AND Boolean bitwise OR OR Boolean bitwise Exclusive OR XOR Complement NOT
	Click to enter a symbol.	N/A
(* *)	Click to enter a comment.	N/A
	n/a	Opens the Block palette.

Select functions from the Block Palette. For detailed information about syntax and operation, see the *InControl Function and Function Block Reference Manual.*

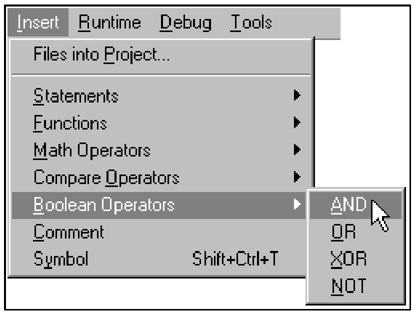


ACOS - Computes the arc cosine of a number in radians

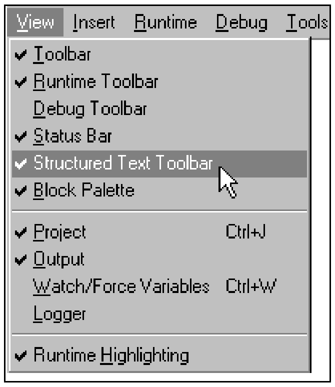
STL Editing Tips

These tips can help as you edit a program.

- You can also enter Structured Text code by making selections from the **Insert** menu, which is shown in the following figure.



- Use the **View** menu to display those objects that you need to see during an editing session. For example, if you prefer to add program elements from the Menu bar, instead of the STL toolbar, you can hide the STL toolbar.



- During an editing session, you can right-click for a fast display of some of the edit options that appear in the menu bar.

Entering Program Code

You can enter the Structured Text code into the edit window by any of these methods.

- Type the lines of code directly into the edit window.
- Click selections on the Structured Text toolbar.
- Click **Insert** on the menu bar and then click the selection.

When you select a block of code and click **Comment** on the **Insert** menu, the code is marked as a comment.

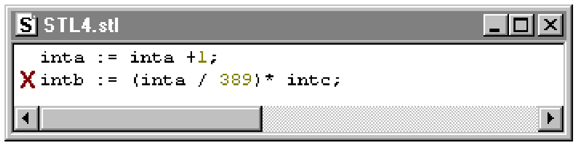
For using symbols in a program, you can choose from these methods:

- If a symbol has not been defined in the Symbol Manager, you can enter a new symbol in the code and then double-click the symbol name. This opens the **Symbol Properties** dialog box, and you can then enter the configuration data for the symbol. The symbol is then automatically added to the Symbol Manager.
- You can double-click anywhere within the Structured Text code and the Symbol Manager opens.

  If you then double-click the name of a symbol appearing in the Symbol Manager, the symbol is inserted in the program at the cursor's location.

  If you double-click an InControl Factory Object (FOE) method, the complete syntax for a function call to the FOE is inserted in the program at the cursor's location.
- You can double-click any symbol name that appears in the program and the Symbol Manager opens. If the symbol has already been defined, the Symbol Manager opens at the correct scoping level with the symbol name selected.

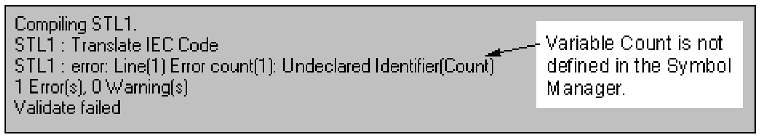
When you validate a program, lines with errors are marked as shown in the following figure.



```
 inta := inta +1;
Xintb := (inta / 389)* intc;
```

STL Validation Errors

You can check the Output window for error messages that can help you troubleshoot the program.



```
Compiling STL1.
STL1 : Translate IEC Code
STL1 : error: Line(1) Error count(1): Undeclared Identifier(Count)
1 Error(s), 0 Warning(s)
Validate failed
```

STL Validation Error Messages

Note The Structured Text editor opens from an SFC Step automatically when you double-click the Step.

Expressions

An expression is defined as a combination of operators (mathematical, logical, or relational) and operands (constants, variables, literal values, or expressions) that can be evaluated, yielding a value of a specific data type, e.g., integer, real number, etc. The operators and data types are described in this section.

Operators

The table below lists the operators that you can use within an expression. The order of precedence determines the sequence in which they are executed within the expression. The operator with the highest precedence is applied first, followed by the operator with the next highest precedence. Operators of equal precedence are evaluated left to right.

Operator	**Symbol**	**Precedence**
Parenthesis	( )	1
Function Evaluation	Identifier (argument list) e.g., LN (A), ABS (X)	2
Exponentiation	**	3
Negate	-	4
Complement	NOT	4
Multiply	*	5
Divide	/	5
Modulus	MOD	5
Add	+	6
Subtract	-	6
Comparison [1]	<, >, <=, >=	7
Equality	=	8
Inequality	<>	8
Boolean/Bitwise AND	AND	9
Boolean/Bitwise Exclusive OR	XOR	10
Boolean/Bitwise OR	OR	11

1 In general, it is recommended that you avoid doing a comparison for equality (or non-equality) with real numbers. If you do this type of comparison using a constant (literal) value and a real variable, the variable must be an LREAL data type to help ensure that you receive the expected result.

These symbols have the following functions:

- = Assigns the value of an expression to a variable (variable:=expression).
- **;** The semicolon is required to designate the end of a statement.
- **[ ]** Brackets are used for array indexing where the array index is an integer. For example, this sets the first element of an array to the value j+3: array[1]: = j + 3;
- (* *) designates a comment. For example, (*This is a comment.*)

Data Types

When an expression is evaluated, the result must be one of the data types supported by InControl. These types are are described in detail in the "Defining Variables" chapter of the *InControl Environment Manual.*

Statement Types

The Structured Text statements, which provide for the actual program execution, consist of the following types:

• Assignment	Sets an object to a specified value.
• BREAK	Causes the program to stop running if you have enabled debugging.
• CASE	Provides for the conditional execution of a set of statements.
• Comment	Provides for comments to be included within the program.
• EXIT	Terminates iterations before the terminal condition becomes TRUE.
• FOR	Indicates that a statement sequence be executed repeatedly based on the value of a control variable.
• Function/Procedure Call	Calls a function or procedure for execution.
• IF	Specifies that one or more statements be executed conditionally.
• INCLUDE	Executes a set of statements contained within an external file.
• REPEAT	Indicates that a statement sequence be executed repeatedly until a Boolean expression evaluates to TRUE.
• RETURN	Used in a function, procedure or a function block to cause program flow to resume in the POU that made the function call.
• SCAN	Causes Structured Text execution to be suspended while an I/O scan is done.

• WHILE	Indicates that a statement sequence be executed repeatedly until a Boolean expression evaluates to FALSE.
• #pragma	Modifies the features and checking operation of the compiler.

In the statement syntax descriptions that follow, the angled brackets < > indicate items for which you substitute a value. Square brackets [ ] indicate items that are optional in the statement. Items that you must type in exactly appear in

`typewriter font.`

Assignment

Use the assignment statement to replace the value of an object with the result of the evaluated expression. The format for the assignment statement is the following:

```
<object> := <expression>;
```

where **<object>** is a variable, array element, etc., and **<expression>** is a single value or expression.

Moving structures and arrays is possible although this type of operation may be lengthy, depending on the size of the structure or array.

To move a structure (Structure1:=Structure2), the size and data types of the structure members must match exactly. No data type conversion is supported for complex moves.

To move an array (Array1:=Array2), the size and data types of the array elements must match exactly. Each element of Array2 is moved to a corresponding element in Array1.

The following examples are Boolean assignment statements. Boolitem1 := TRUE; Boolitem2 := (val <= 75);

The following example sets an element in an array to the value resulting from the evaluation of a real number expression.

RealArray[13] := (rla / rlb)* 13.41574;

The following example sets a string variable to the value of a string. Be sure to enclose the string in single quotation marks.

String_Val := ‘This is a string literal’;

The following example assigns a value to the process variable for the PID

InControl factory object (FOE) named BoilerTempControl. BoilerTempControl.PV := 500.0;

BREAK

The BREAK statement causes program flow to stop and is useful for debugging a program. To enable BREAK statements used in a program, click **Enable Debug** on the **Validate Project** or **Validate Program** dialog boxes or on the **Properties** dialog box for the program. For Structured Text programs, consider using the breakpoint instead. See "Using Breakpoints" in the "Running a Project" chapter of the *InControl Environment Manual.* The format for the BREAK statement is the following:

```
BREAK;
```

The following is an example of the BREAK statement.

```
intout := BCD_TO_INT(bcd_in);

BREAK;

Max_num := MAX (num1, num2);
```

Note The following statements cannot appear on the same line: BREAK, SCAN, END_FOR, END_FOR_NOWAIT, END_WHILE, and END_WHILE_NOWAIT.

CASE

Use the CASE statement to design for the execution of a set of statements based on the value of a variable. The construction of the CASE statement consists of the following.

- An expression that evaluates to a value of data type ANY_INT.
- A list of statement groups, with each group labeled by one or more integers or ranges of integers. You can also use an enumeration or a symbol defined as a constant for the label.

When the label for a set of statements matches the value of the ANY_INT expression, the statements in that set are executed. If the label consists of more than one integer (enumeration, constant), or a range, the match can occur with any of the integers (enumerations, constants) contained in the label. If no match occurs and you have included an ELSE statement, the statements following the ELSE are executed. If there is no ELSE statement, no statements are executed in the CASE statement.

The format for the CASE statement is the following:

```
CASE <expression> OF
    label: BEGIN <statement list> END
    label,label,label: BEGIN <statement list> END
    label..label: BEGIN <statement list> END

ELSE
    <statement list>

END_CASE;
```

where <**expression**> evaluates to an ANY_INT data type, **label** is an ANY_INT literal value, enumeration or constant symbol, and **statement list** is any set of valid Structured Text statements.

The following example selects a new string to display.

```
CASE ASelection OF
   0: OutputString := 'Dave';
   1: OutputString:='Tim';
   2,3,4: OutputString:='Steve';
   5..9: OutputString:='Alan';
ELSE
   OutputString:='Empty';
END_CASE;
```

In the following example, the label is an enumeration, of type COLORS with these members: BLUE, GREEN RED, YELLOW.

```
CASE COLORS OF
   COLORS.BLUE: BEGIN OutputString := 'BLUE'; END
   COLORS.GREEN: BEGIN OutputString:='GREEN'; END
   COLORS.RED: BEGIN OutputString:='RED'; END
   COLORS.YELLOW: BEGIN OutputString:='YELLOW' ; END
ELSE
   (*Error*)
END_CASE;
```

Comment

Use the COMMENT statement to incorporate useful annotations into your program code. The format for a comment is the following:

```
(* <free-form text> *)
```

The following is an example of a comment statement.

```
(* Select a new string to display*)
CASE lSelection OF
   0:
(* This selects "Dave" as the output*)
      OutputString := 'Dave';
   1:
(* This selects "Shiela" as the output*)
      OutputString:='Shiela';
END_CASE;
```

You can also comment more than one line at a time:

```
(* Set intvarA to 3
   Then set intvarB to 4 *)
```

Note If you embed a comment within a comment, a compiler error occurs.

EXIT

Use the EXIT statement to terminate an iterative process, e.g. a FOR or WHILE statement, before the normal termination of the process. The format for the statement is the following:

```
<condition for exiting> EXIT;
```

where **`<condition for exiting>`** is an expression that determines whether to terminate early.

If you use the EXIT statement within a nested iteration, the exit occurs from the loop in which the EXIT is located, and program flow resumes after the statement that normally ends the iteration, e.g., END_FOR, END_WHILE, etc.

The following example shows the operation of the EXIT statement. When the variable called cancel equals 0, then the variable called tally equals 0; when the variable called cancel does not equal 0, tally equals 46.

```
tally:=0;

FOR counta := 1 TO 4 DO
      FOR countb := 1 TO 3 DO
         FOR countc := 1 TO 2 DO
         IF cancel = 0 THEN EXIT; END_IF;
         tally := tally + countc;
         END_FOR;
      END_FOR;
   tally := tally + counta;

END_FOR;
```

Note Typically, well-designed loops do not require EXIT statements. Use EXIT statements sparingly. To avoid a potentially endless loop condition, be sure that all conditions for completing any nested iterations are satisfied after an EXIT is executed.

FOR

Use the FOR statement to execute a series of statements repeatedly, with the number of repetitions based on the value of a control variable. Each iteration changes the value of the control variable (the default is 1), which can be an expression, and which follows the BY portion of the statement. The control variable can be positive or negative. The FOR statement checks the control variable before each iteration and the statements within the FOR/END_FOR boundary are not executed when the current value of the control variable has reached (or exceeded) the limit.

The format for the FOR statement is the following:

```
FOR <INT variable> := <expression> TO <expression> [BY
    <expression>]
DO
    <statement list>
END_FOR;
```

where **<INT variable>** is an ANY_INT data type, **<expression>** resolves to an ANY_INT data type, and **<statement list>** is any set of valid Structured Text statements.

To help avoid the possibility of designing a loop that does not end, it is suggested that you not change the value of the loop control variable within the loop.

The END_FOR statement causes the system to wait for an I/O scan at the end of every cycle of the FOR loop. As an enhancement to the IEC-61131-3 specification, you can use the END_FOR_NOWAIT statement to loop back without continuing the other tasks on the timeline (the I/O and other programs are not scanned). You can use the "EXIT" statement to end a FOR before its normal termination.

WARNING! If the END_FOR_NOWAIT statement causes an endless loop condition, the Watchdog Timeout will expire. This causes the runtime engine service to shut down, stopping all programs, with the potential risk of death or injury to personnel and/or damage to equipment. Design and test your code to verify that an endless loop condition does not occur. If you use an EXIT statement within a nested iteration, be sure that all conditions for completing the iteration are satisfied to avoid a potentially endless loop condition.
Do not use a TMR variable <TMR name>.Q in the FOR condition section if you also use the END_FOR_NOWAIT statement. This variable is not processed until the FOR loop is finished, and the loop cannot finish executing until the variable is processed.

Note The following statements cannot appear on the same line: BREAK, SCAN, END_FOR, END_FOR_NOWAIT, END_WHILE, and END_WHILE_NOWAIT.

The following example shows the operation of the FOR statement.

```
total:=0;

FOR count:= 1 TO 100 DO
    total:= total + 1;

END_FOR;

total:=0;

FOR t := 10 TO 1 BY -1 DO
    total := total + 5;

END_FOR;
```

Function/Procedure Call

The Structured Text function or procedure call executes one of the InControl predefined functions or function blocks. You can also use the function or procedure call to execute FOE methods and user-defined functions or function blocks.

A function that returns a value operates as a true function and you use it on the right side of an Assignment statement. The format for a function call is the following:

```
<result>:= <function name> (parameter_1, parameter_2, . .
    .);
```

The following example shows the syntax of the TAN function call.

```
TrigAnswer := TAN (input);
```

A function that does not return a value operates like a procedure. The format for a procedure call is the following:

```
<procedure name> (parameter_1, parameter_2, . . .) ;
```

The following example shows the syntax of the OPENFILE procedure:

```
OPENFILE (FCB:= <fcb>, FILE:= <filename>);
```

If a parameter is not the correct data type, a validation error occurs.

For more information about designing a user-defined function or function block, see the "Project Organization and Management" chapter of the *InControl Environment Manual.*

All FOEs, which do not have a method scheduled to execute automatically, and all functions and function blocks enter the Loaded mode when you download them to the runtime engine. An FOE, function, or function block that is in the Loaded mode, has been loaded in the runtime engine and runs when called for execution.

IF

Use the IF statement to design the execution of a set of statements only when a Boolean variable or expression is TRUE. The construction of the IF statement consists of the following.

- A Boolean expression preceded by the IF, and followed by THEN, that causes a set of statements to execute when it is TRUE.
- A second (optional) Boolean expression preceded by ELSEIF, and followed by THEN, that causes a second set of statements to execute if the first set does not execute and the second condition is TRUE.
- A set of statements preceded by ELSE (optional) that is executed if the IF and ELSEIF statements do not execute.
- An END_IF statement is required to close the IF statement.
- If neither Boolean expression is TRUE and you have included an ELSE statement, the statements following the ELSE are executed. If there is no ELSE statement, no statements are executed in the IF construction.

The format for the IF statement is the following:

```
IF <Boolean expression> THEN
   <statement list>

[ELSEIF <Boolean expression> THEN
   <statement list> ]

[ ELSE
   <statement list> ]

END_IF;
```

where **<Boolean expression>** is any expression that evaluates to a Boolean value, and **<statement list>** is any set of valid Structured Text statements.

The following is an example of the IF statement usage.

```
com_value := com_input_BUFF[11];

IF COM_VALUE = 11 THEN
   lSelection := lSelection + 1;

ELSEIF COM_VALUE = 10 THEN
   lSelection:= lSelection -1;

ELSE
   lSelection:= 0;

END_IF;
```

INCLUDE

Use the INCLUDE statement to call an external file and execute a set of statements contained within the file.

The format for the INCLUDE statement is the following:

```
INCLUDE ‘ < string > ’;
```

where **<string>** specifies the path and file containing the statements to be executed.

You can use more than one INCLUDE statement if you need to call multiple files. Program commands can be mixed with the INCLUDE statements.

To simplify code debugging and maintenance, it is suggested that you not use INCLUDE statements in your programs.

The following is an example of the INCLUDE statement usage.

```
InCLUDE ‘C:\ST_FILES\STFILE1.TXT’;
```

REPEAT

Use the REPEAT statement to design the repeated execution of a set of statements until a Boolean condition becomes TRUE. Statements always execute at least one time before the Boolean expression is evaluated.

The END_REPEAT statement causes the system to do an I/O scan at the end of every cycle of the REPEAT loop. As an enhancement to the IEC-61131-3 specification, you can use the END_REPEAT_NOWAIT statement to loop back without continuing the other tasks on the timeline (the I/O and other programs are not scanned). You can use the "EXIT" statement to end a REPEAT before its normal termination.

WARNING! If the END_REPEAT_NOWAIT statement causes an endless loop condition, the Watchdog Timeout will expire. This causes the runtime engine service to shut down, stopping all programs, with the potential risk of death or injury to personnel and/or damage to equipment. Design and test your code to verify that an endless loop condition does not occur. If you use an EXIT statement within a nested iteration, be sure all conditions for completing the iteration are satisfied to avoid a potentially endless loop condition.
Do not use a TMR variable <TMR name>.Q in the REPEAT condition section if you also use the END_REPEAT_NOWAIT statement. This variable is not processed until the REPEAT loop is finished, and the loop cannot finish executing until the variable is processed.

The format for the REPEAT statement is the following:

```
REPEAT
       <statement list>

UNTIL  <Boolean expression>   END_REPEAT;
```

where **<Boolean expression>** is any expression that resolves to a Boolean value, and **<statement list>** is any set of valid Structured Text statements.

Note The following statements cannot appear on the same line: BREAK, SCAN, END_FOR, END_FOR_NOWAIT, END_WHILE, and END_WHILE_NOWAIT.

The following is an example of the REPEAT statement usage.

```
ZZ := 0;

REPEAT
   ZZ := ZZ+5;

UNTIL   ZZ >= 100 END_REPEAT;
```

RETURN

Use a RETURN in a function, procedure, or function block to cause program flow to resume in the POU that called the function, procedure, or function block for execution.

The format for the RETURN statement is the following:

```
RETURN;
```

SCAN

Use the SCAN statement to suspend the execution of Structured Text statements until after the next I/O scan. The format for the SCAN statement is the following:

```
SCAN;
```

Note The following statements cannot appear on the same line: BREAK, SCAN, END_FOR, END_FOR_NOWAIT, END_WHILE, and END_WHILE_NOWAIT.

The following is an example of the SCAN statement usage.

```
ASelection := ASelection + 1;

SCAN;

IF ASelection >= 3 THEN
   ASelection := 0;

ELSEIF ASelection < 0 THEN
   ASelection := 2;

END_IF;
```

WHILE

Use the WHILE statement to design the repeated execution of a set of statements until a Boolean condition becomes FALSE. If the Boolean condition is initially FALSE, then the statements are not executed at all.

The END_WHILE statement causes the system to do an I/O scan at the end of every cycle of the WHILE loop. As an enhancement to the IEC-61131-3 specification, you can use the END_WHILE_NOWAIT statement to loop back without continuing the other tasks on the timeline (the I/O and other programs are not scanned). You can use the "EXIT" statement to end a WHILE before its normal termination.

WARNING! If the END_WHILE_NOWAIT statement causes an endless loop condition, the Watchdog Timeout will expire. This causes the runtime engine service to shut down, stopping all programs, with the potential risk of death or injury to personnel and/or damage to equipment. Design and test your code to verify that an endless loop condition does not occur. If you use an EXIT statement within a nested iteration, be sure that all conditions for completing the iteration are satisfied to avoid a potentially endless loop condition.
Do not use a TMR variable <TMR name>.Q in the WHILE condition section if you also use the END_WHILE_NOWAIT statement. This variable is not processed until the WHILE loop is finished, and the loop cannot finish executing until the variable is processed.

The format for the WHILE statement is the following:

```
WHILE   <Boolean expression>   DO
   <statement list>

END_WHILE;
```

where **<Boolean expression>** is any expression that resolves to a Boolean value, and **<statement list>** is any set of valid Structured Text statements.

Always place a semicolon after the END_WHILE statement and each of the statements in the statement lists.

Note The following statements cannot appear on the same line: BREAK, SCAN, END_FOR, END_FOR_NOWAIT, END_WHILE, and END_WHILE_NOWAIT.

The END_WHILE statement cannot be used in functions or function blocks. Use END_WHILE_NOWAIT to complete the WHILE statement.

The following is an example of the WHILE statement usage.

```
VAL_A := 1;

WHILE (VAL_A <= 100) DO
   VAL_A := VAL_A*5;

END_WHILE;
```

#pragma

Use the #pragma statement to change the form of the message issued by the compiler when a program is validated and there are one or more instances of writing an output value to an I/O symbol that has been defined as an input. The format for the #pragma statement is the following:

```
#pragma <parameter>
```

where <parameter>

can be one of these values: IOWriteError, IOWriteWarn, or IOWriteIgnore.

A #pragma statement must start in the first column, be the only item on the line, and does not take a semicolon. The #pragma statement affects only the program in which it is used.

The following is an example of the #pragma statement usage. A warning is generated if the program includes an instance of writing an output value to an I/O input.

```
#pragma IOWriteWarn
```

The following is another example of the #pragma statement usage. No message is generated if the program includes an instance of writing an output value to an I/O input.

```
#pragma IOWriteIgnore
```

For information about modifying the compiler messaging system so that the programs in all projects are affected see "Displaying Compiler Warnings" in the "InControl System Administration" chapter of the *InControl Environment Manual.*

For information about modifying the compiler messaging system so that the programs in all projects are affected, see "Displaying Compiler Warnings" in the "InControl System Administration" chapter of the *InControl Environment Manual.*

InControl Functions and Function Blocks

InControl RLL programs support predefined and user-defined functions and function blocks.

The predefined functions and function blocks supported by InControl are listed in the following table. For information about syntax and operation, see the *InControl Function and Function Block Reference Manual.*

Functions/Procedures by Group

Group	Type	Description
Bitwise	AND	Computes the bitwise AND of two numbers.
	NOT	Computes the bitwise complement of a number.
	OR	Computes the bitwise OR of two numbers.
	ROL	Rotates the input left by a specified number of bits.
	ROR	Rotates the input right by a specified number of bits.
	SHL	Shifts the input left by a specified number of bits.
	SHR	Shifts the input right by a specified number of bits.
	XOR	Computes the bitwise Exclusive OR of two numbers.
Comparison	EQ	Tests two inputs for equality.
	GE	Tests if first input is greater than or equal second input.
	GT	Tests if first input is greater than second input.
	LE	Tests if first input is less than or equal second input.
	LT	Tests if first input is less than second input.
	NE	Tests two inputs for inequality.

Group	Type	Description
Conversion	ARRAY_TO_STRING	Takes a byte array input and stores the bytes as characters in a string.
	BCD_TO_INT	Converts a Binary-Coded Decimal (BCD) input to an ANY_INT value.
	DATE_TO_REAL	Converts a DATE data type input to an ANY_REAL value.
	DATE_TO_STRING	Converts a DATE data type input to a string.
	INT_TO_BCD	Converts an integer to the equivalent Binary-Coded Decimal (BCD) representation of the value.
	INT_TO_REAL	Converts an ANY_INT input to an ANY_REAL value.
	INT_TO_STRING	Converts an ANY_INT input to a string.
	REAL_TO_DATE	Converts an ANY_REAL input to a DATEvalue.
	REAL_TO_INT	Converts an ANY_REAL input to an ANY_INT value.
	REAL_TO_STRING	Converts an ANY_REAL input to a string.
	REAL_TO_TIME	Converts an ANY_REAL input to a TIMEvalue.
	STRING_TO_ARRAY	Takes a string input and stores the characters of the string in a byte array.
	STRING_TO_DATE	Converts an input string to a DATE value.
	STRING_TO_INT	Converts an input string to an ANY_INT value.
	STRING_TO_REAL	Converts a string input to an ANY_REAL value.
	STRING_TO_TIME	Converts a string input to a TIME value.
	TIME_TO_REAL	Converts a TIME input to an ANY_REAL value.
	TIME_TO_STRING	Converts a TIME input to a string.
Counter	CTD	Counts events by decrementing by one.
	CTU	Counts events by incrementing by one.
	CTUD	Counts events up or down.

Group	Type	Description
File	CLOSEFILE	Closes a file.
	COPYFILE	Copies a file.
	DELETEFILE	Deletes a file.
	NEWFILE	Creates a new file.
	OPENFILE	Opens an existing file.
	READFILE	Reads data from a file.
	REWINDFILE	Rewinds a file to the beginning.
	WRITEFILE	Writes data to a file.
Math	ABS	Computes the absolute value of a value.
	ADD	Adds two values.
	DIV	Divides one value by another.
	EXPT	Raises a value to the power specified by a second value.
	MAX	Determines the larger of two values.
	MIN	Determines the smaller of two values.
	MOD	Divides one value by another and stores the remainder.
	MOVE	Copies data from one location to another.
	MUL	Multiplies two values.
	NEG	Negates (inverts) the inputs.
	SQRT	Computes the square root of a value.
	SUB	Subtracts one value from another.
	TRUNC	Removes one or more of the least significant digits of an ANY_REAL data type.

Group	Type	Description
String	CONCAT	Concatenates a string input to the end of another string.
	DELETE	Deletes characters from the middle of a string input.
	FIND	Searches for one string input within another.
	INSERT	Inserts a string input into another string.
	LEFT	Copies the leftmost characters from a string input.
	LEN	Stores the length of a string input.
	MID	Copies characters from the middle of a string input.
	MSGWND	Displays a message in the Output Window.
	REPLACE	Replaces characters in a string input with another string input.
	RIGHT	Copies the rightmost characters from a string input.
Timer	TOF	Provides off-delay timing of events.
	TON	Provides on-delay timing of events.
	TP	Activated by a pulse, provides off-delay timing of events.
Trig/Log	ACOS	Computes the arc cosine of a value.
	ASIN	Computes the arc sine of a value.
	ATAN	Computes the arc tangent of a value.
	COS	Computes the cosine of a value.
	EXP	Computes the natural log exponentiation of a value.
	LN	Computes the natural log of a value.
	LOG	Computes the log (base 10) of a value.
	SIN	Computes the sine of a value.
	TAN	Computes the tangent of a value.
Trigger	ABORT_ALL	Aborts all programs that are running.
	F_TRIG	Turns on an output when triggered by a falling edge trigger.
	R_TRIG	Turns on an output when triggered by a rising edge trigger.

APPENDIX A

RLL Example Program

This appendix presents examples for how to design a simple RLL program.

Contents

Developing an RLL Program

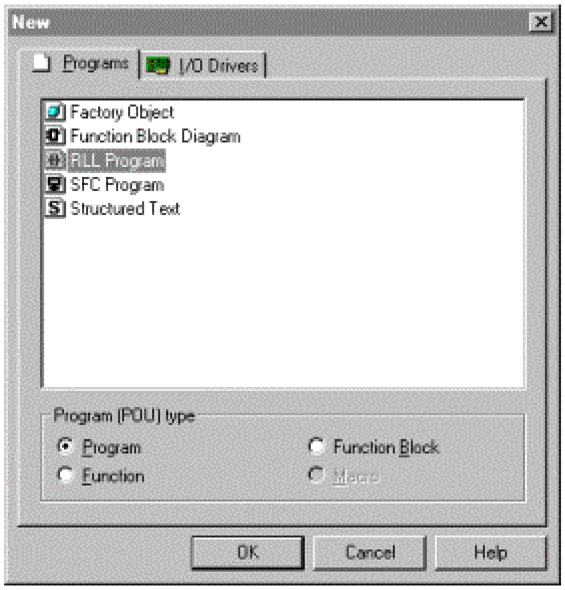
This section describes how to create a new RLL program.

Creating a New RLL Program

To begin developing an RLL program:

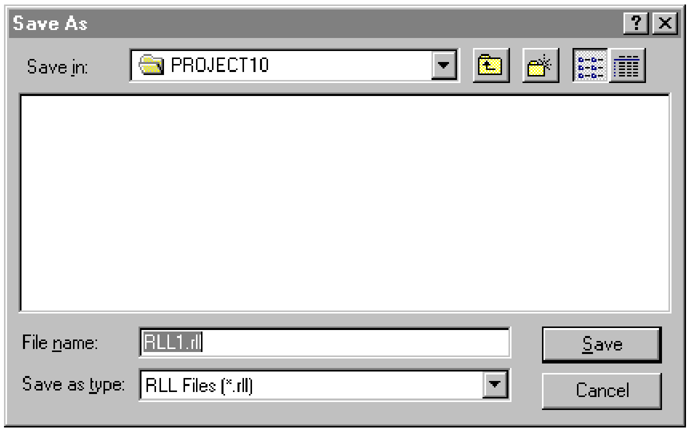
1. On the File menu, click New.

   The New dialog box appears.



2. Click RLL Program.
3. Be sure that Program is the selected Program Type. Then click OK.

   The Save As dialog box appears.



4. Choose the default or a new name (up to 31 characters) for the program and click **Save.** The default extension .rll is appended when the file is saved in your project.

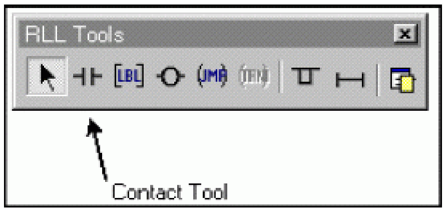
The RLL editor displays a new RLL file with the two power rails and an empty rung. To begin editing the program, start by adding a contact. Follow the procedure "Adding a Contact."

For a detailed description of the RLL editor and toolbar items, see the "Using the RLL Editor" chapter.

Adding a Contact

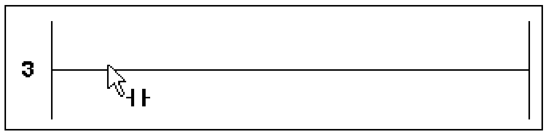
To add a contact to the rung:

1. Click **Contact Tool** on the RLL toolbar.

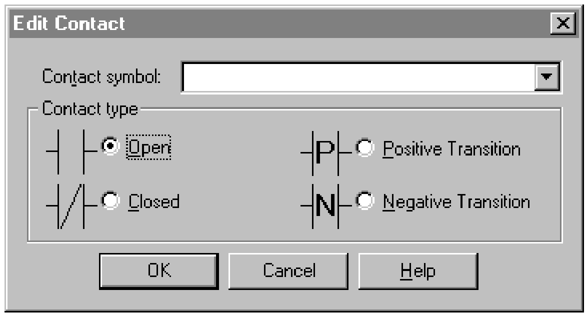


Note that the RLL toolbar may be docked and in a vertical orientation.

2. Move the cursor over the rung.



3. Click the left mouse button. The **Edit Contact** dialog box appears.


Edit Contact

Contact symbol:

Contact type

Open [✓]

Closed

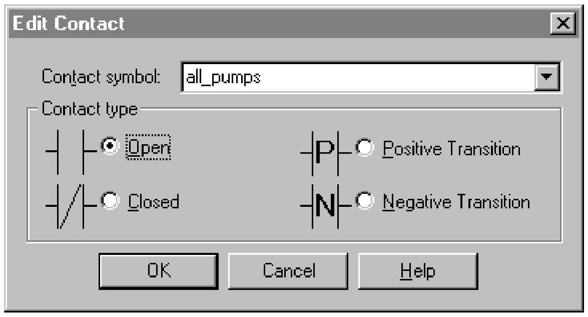
Positive Transition

Negative Transition

OK Cancel Help

RLL Example: Edit Contact Dialog Box 1

4. Enter a name in the **Contact Symbol** field. Only the alphanumeric characters and the underscore character are valid. Do not use spaces and do not begin the name with a number. Symbol names are not case sensitive. In the figure, all_pumps is the new variable.



Contact symbol: all_pumps

Contact type: [✓ Open] Closed

RLL Example: Edit Contact Dialog Box 2

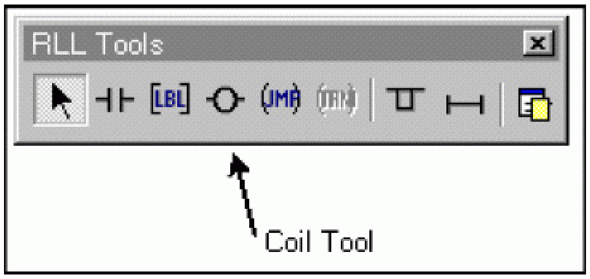
5. Click **Open** to specify the type of contact and then click **OK.**
6. When the system prompts you, click **Add Global** to add the new variable name to the Symbol Manager as a global variable. This allocates an internal memory location to represent the new contact.

For more information about defining variables, see "Creating a Variable" in the "Defining Variables" chapter.

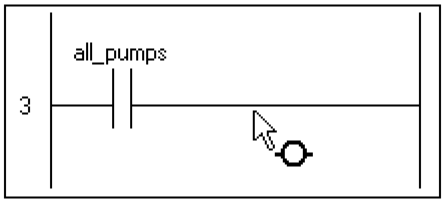
Adding a Coil

To add a coil to the rung:

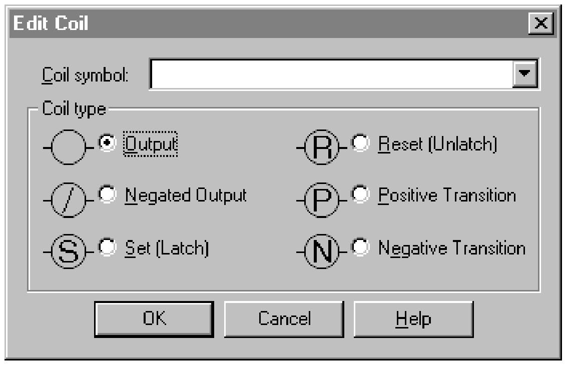
1. Click **Coil Tool** on the RLL toolbar.



2. Move the cursor to the location on the rung to the right of the contact.



3. Click the left mouse button. The **Edit Coil** dialog box appears.



RLL Example: Edit Coil Dialog Box 1

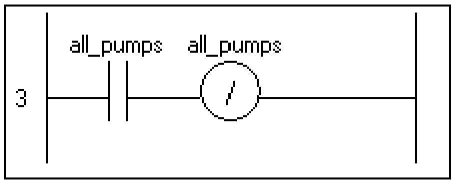
4. Because this example uses the same variable for both the contact and the new coil, it is not necessary to add another variable to the Symbol Manager. Select all_pumps as the variable name for the coil.

5. Click **Negated Output** to specify the type of coil. Then click **OK.** The new coil, a Negated Output Coil, appears on the rung.

   The asterisk by the program name in the window title bar indicates that the program has not been saved; or when the RLL is executing on the runtime engine, that this is not the same version because editing changes have been made.

6. On the **File** menu click **Save** to save your work.

The following figure shows the program after you have entered both program elements:



RLL Example: New Contact and Coil

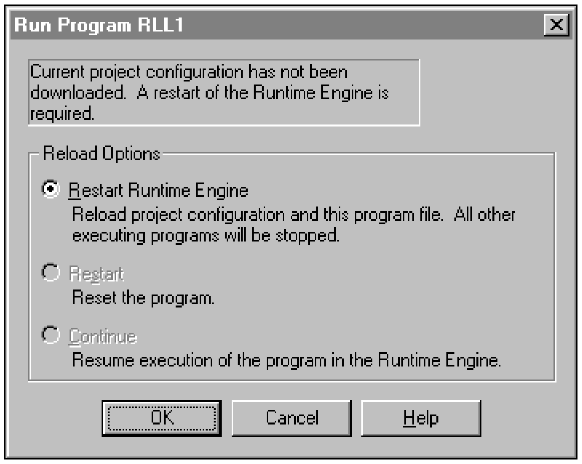
To validate, download, and run the program, see "Running the RLL Program."

Running the RLL Program

This section describes how to run the RLL program.

To run the RLL program:

1. On the **Runtime** menu, click **Connect.** This connects the Development environment to the runtime engine.
2. On the **Runtime** menu, click **Run Program.** The **Run Program** dialog box appears.



Run Program RLL1

Current project configuration has not been downloaded. A restart of the Runtime Engine is required.

Reload Options

[✓ Restart Runtime Engine]
Reload project configuration and this program file. All other executing programs will be stopped.

Restart
Reset the program.

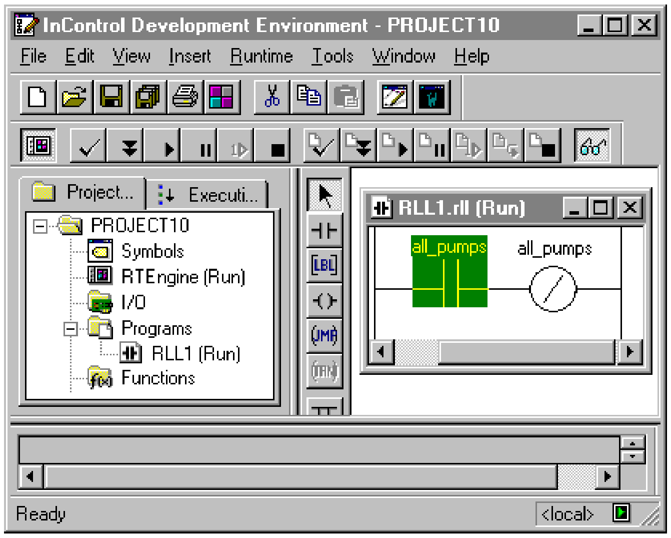
Continue
Resume execution of the program in the Runtime Engine.

RLL Example: Run Program Dialog Box

3. Click **OK** to restart the runtime engine.

   The program is compiled, downloaded to the runtime engine, and begins running. The runtime highlighting shows each program element as it is executed.

Note If the program elements are not highlighted, on the **View** menu, click **Runtime Highlighting.**



RLL Example: Running the Program

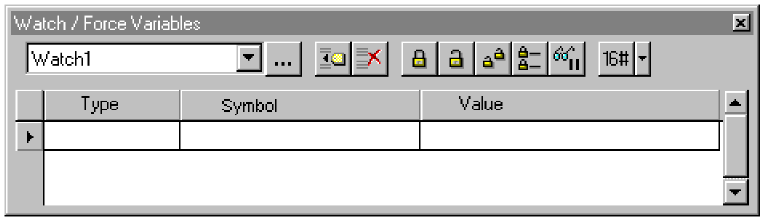
Verify the processes by checking the Output window at the bottom of the screen.

Monitoring Variables in the RLL Program

This section describes how to observe the values of the variables in the RLL program.

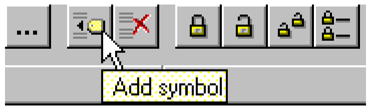
To monitor the RLL program variables:

1. If the Watch window is not visible, on the **View** menu, click **Watch/Force Variables.** The Watch window appears.

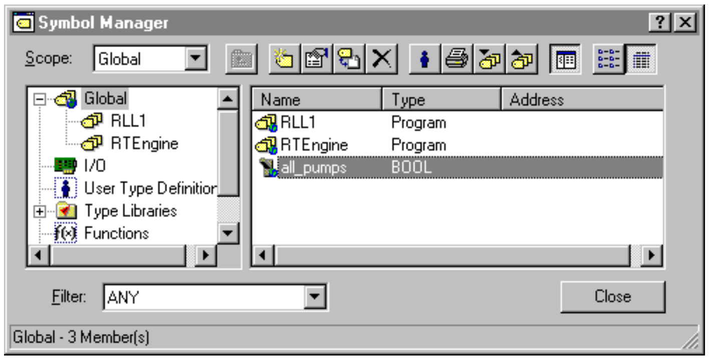


RLL Example: Watch Window 1

2. To add the variable all_pumps to the Watch window, click **Add Symbol.**



The **Symbol Manager** dialog box appears. Since the all_pumps variable is global, it appears under the Global category of variables.



Symbol Manager

Scope: Global

Global
RLL1
RTEngine
I/O
User Type Definitior
Type Libraries
Functions

Name	Type	Address
RLL1	Program	
RTEngine	Program	
all_pumps	BOOL	

Filter: ANY

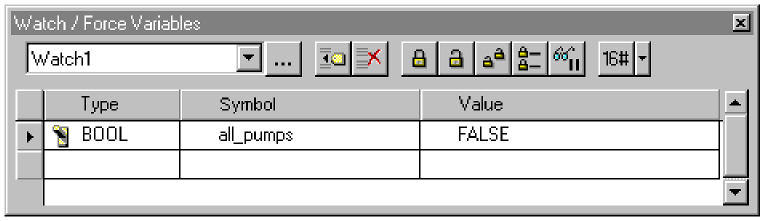
Close

Global - 3 Member(s)

RLL Example: Symbol Manager

3. Click the variable name to select it, then click **OK.**

   The variable all_pumps is added to the Watch window, and its value is updated as its status changes.



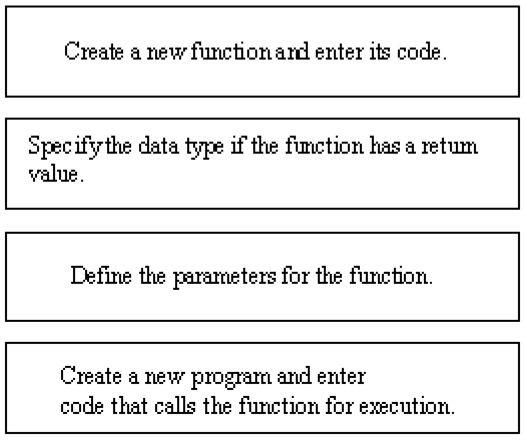
Type	Symbol	Value
BOOL	all_pumps	FALSE

RLL Example: Watch Window 2

Add other program elements to the RLL program to become familiar with the rest of the RLL programming editor.

Developing a Function

This section describes how to create a new function. The following is the general procedure that you will follow:


Create a new function and enter its code.
Specify the data type if the function has a return value.
Define the parameters for the function.
Create a new program and enter code that calls the function for execution.

Note The figures in this section are based on a new project with no other programs. If you have already created an RLL program, for example, you may observe some minor differences in the dialog boxes.

Creating a New Function

To begin developing a function:

1. On the File menu, click New.

   The **New** dialog box appears.

2. Click **RLL Program.**

3. Click **Function** to select the Program Type. Then click **OK.**

   The Save As dialog box appears.

4. Enter a name for the function (up to 31 characters) and click Save.This example uses calc_addR.rll for the name. The default extension .rll is appended when the file is saved in your project.

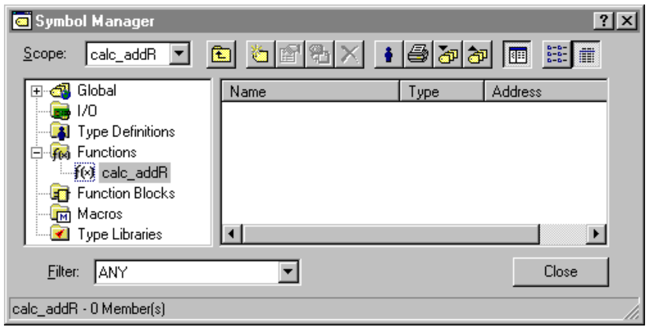
   The RLL editor displays a new RLL file ready for editing. To begin editing the function, see "Entering Function Code."

Specifying Return Value Data Type

You specify the return type for a function in the Symbol Manager.

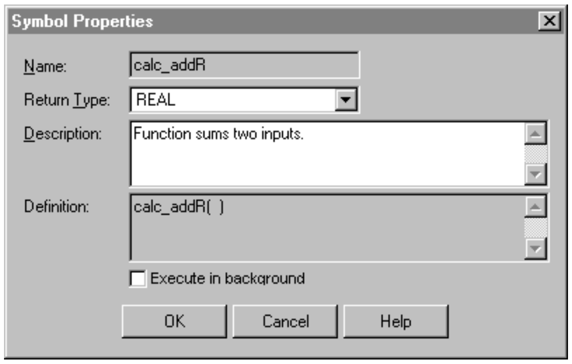
To specify the return data type for the function:

1. On the **Tools** menu, click **Symbol Manager.** The Symbol Manager appears.



calc_addR - 0 Member(s)

2. Right-click the function and select **Properties.** The **Symbol Properties** dialog box appears.



Name: calc_addR
Return Type: REAL
Description: Function sums two inputs.
Definition: calc_addR( )
☐ Execute in background

RLL Function Example: Return Value Properties

3. Select REAL as the data type in the **Return Type** field and enter an optional description for the function.

   Functions only return simple data types. They cannot return arrays, structures, or function blocks.

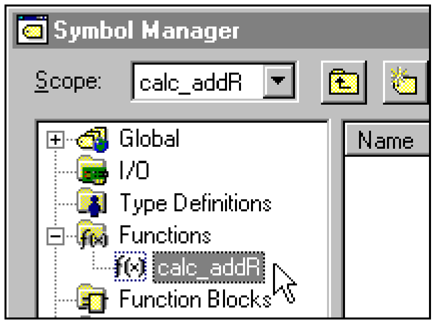
4. Click **OK** to save your work.

Creating Function Parameters

You define the input and output parameters and variables for a function in the Symbol Manager. Function variables are local to the function and cannot be referenced elsewhere in the project, except within the context of the function call. Within the function, input parameters are read only. Output parameters must be assigned values through an assignment statement.

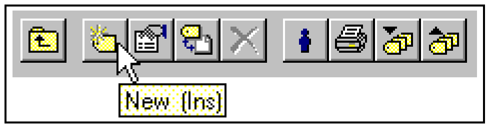
To define the parameters for the function:

1. On the **Tools** menu, click **Symbol Manager.**
2. Click the function to select it as shown in the following figure.

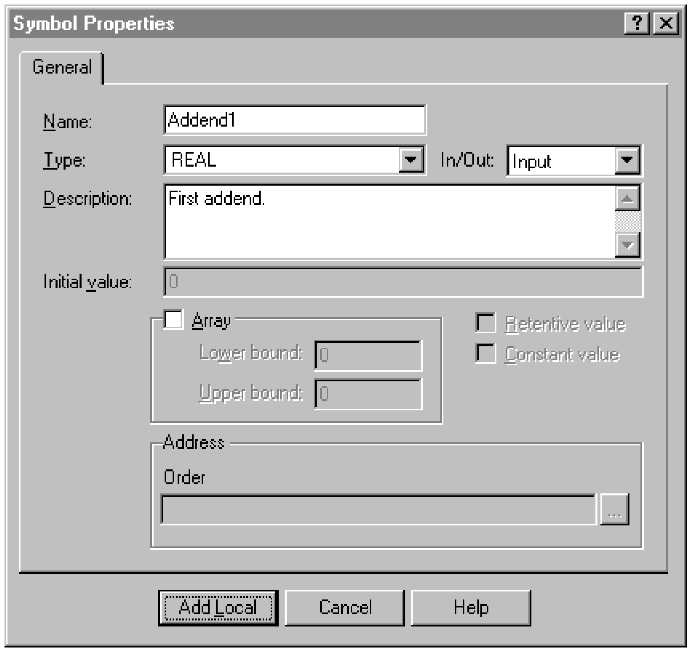


RLL Function Example: Selecting the Function

3. Click **New** on the Symbol Manager toolbar.



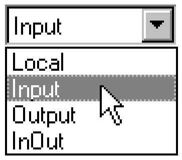
The **Symbol Properties** dialog box appears.



Name: Addend1
Type: REAL
In/Out: Input
Description: First addend.
Initial value: 0
Array
Lower bound: 0
Upper bound: 0
Retentive value
Constant value
Address
Order

4. Enter addend1 for the name of the first parameter into the **Name** field.
5. Select REAL as the data type in the **Type** field.
6. Enter the optional description into the **Description** field. This example uses the following text:

   First addend.
7. Choose Input in the **In/Out** field, as shown below.

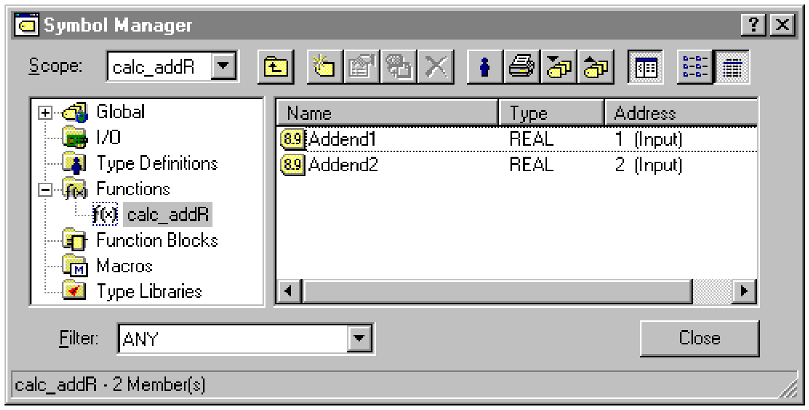


8. Click **Add Local** to complete the definition for this parameter.
9. Before closing the Symbol Manager, repeat steps 3-8 to add the other input parameter used by the example function block. Both are listed in the table below. Enter them in the order shown.
10. Click **Close** to close the Symbol Manager.

The parameters used by the function are listed in the following table.

Name	Data Type	In/Out	Description
Addend1	REAL	Input	First addend.
Addend2	REAL	Input	Second addend.

The following figure shows the contents of the Symbol Manager after the two parameters have been defined. Although it does not matter for this example function, which only adds two numbers, the order of the parameters in the Symbol Manager is important. Parameter order is indicated in the **Address** field, as shown below.



Name	Type	Address
Addend1	REAL	1 (Input)
Addend2	REAL	2 (Input)

calc_addR - 2 Member(s)

RLL Function Example: Parameters

If you enter parameters out of order, you can change their order in the Symbol Manager.

To change the order of parameters:

1. Right-click the parameter.

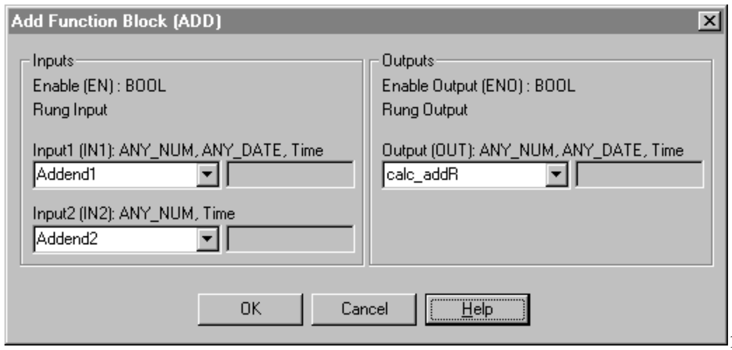
2. Click **Decrease Address** or **Increase Address** to change the order of the parameter in the list.

Entering Function Code

To enter the code for the function, follow the steps below. You can copy the code from the online manual or help, paste it directly into the editor window, and avoid typing it in manually.

To enter the example code:

1. Place an Add function on the first rung. Use the variable names shown in the following figure.


Add Function Block (ADD)

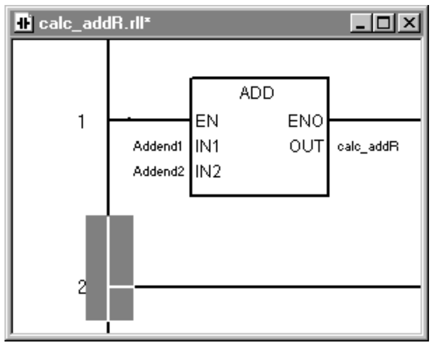
Inputs
Enable (EN) : BOOL
Rung Input
Input1 (IN1): ANY_NUM, ANY_DATE, Time
Addend1
Input2 (IN2): ANY_NUM, Time
Addend2

Outputs
Enable Output (ENO) : BOOL
Rung Output
Output (OUT): ANY_NUM, ANY_DATE, Time
calc_addR

m

RLL Function Example: Entering Code 1

The following figure shows the contents of the RLL editor after the Add function is entered:

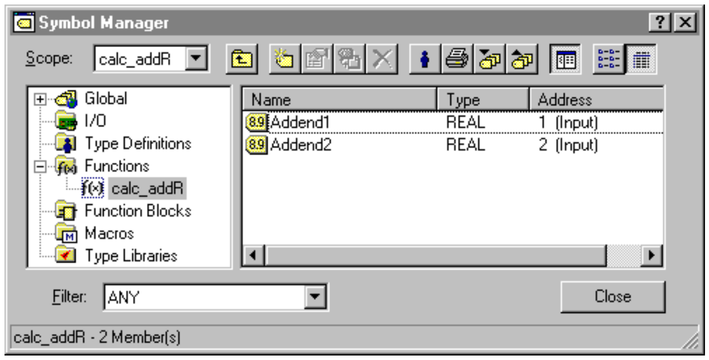


RLL Function Example: Entering Code 2

The asterisk by the program name in the window title bar indicates that the program has not been saved; or when the program is executing on the runtime engine, that this is not the same version because editing changes have been made.

2. On the **File** menu, click **Save.**

The following figure shows the contents of the Symbol Manager after the two parameters have been defined. Although it does not matter for this example function, which only adds two numbers, the order of the parameters in the Symbol Manager is important. Parameter order is indicated in the Address field, as shown below.



Name	Type	Address
Addend1	REAL	1 (Input)
Addend2	REAL	2 (Input)

calc_addR - 2 Member(s)

RLL Function Example: Parameters

If you enter parameters out of order, you can change their order in the Symbol Manager.

To change the order of parameters:

3. Right-click the parameter.
4. Click **Decrease Address** or **Increase Address** to change the order of the parameter in the list.

Creating the Calling Program

A function does not run automatically, but rather must be called by a program. This section describes how to create an RLL program to call the example function.

To create an RLL Program:

1. Follow the procedure that you used to create a function, as described in "Creating a New Function."

   On the **File** menu, click **New.**

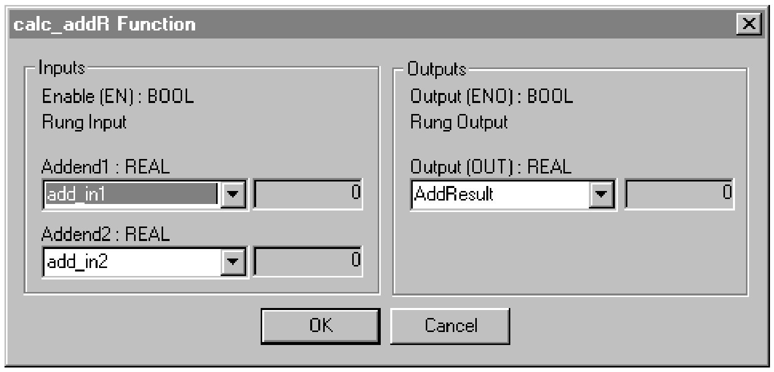
   Click **RLL Program.**

   Click **Program** to select the Program Type. Then click **OK.**

   Enter a name for the program (up to 31 characters) and click **Save**.This example uses test_functionR.rll for the name.

   The RLL editor displays a new RLL file ready for editing.

2. Open the Function palette and place the new user-defined function calc_addR on the first rung. Use the variable names shown in the following figure.



calc_addR Function

Inputs
Enable (EN) : BOOL
Rung Input

Addend1 : REAL
add_in1 0

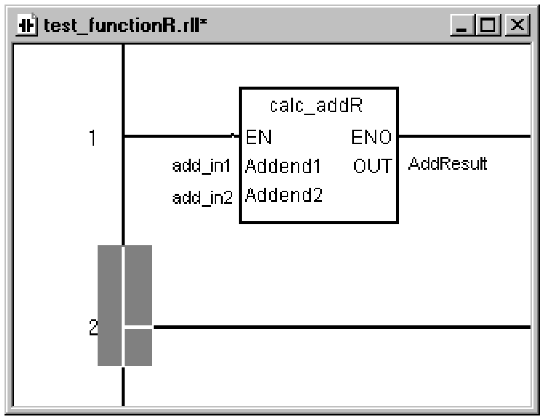
Addend2 : REAL
add_in2 0

Outputs
Output (ENO) : BOOL
Rung Output

Output (OUT) : REAL
AddResult 0

RLL Function Example: Code for Calling Program 1

The following figure shows the contents of the RLL editor after the Calc_addR user-defined function is entered:



RLL Function Example: Code for Calling Program 2

3. On the **File** menu, click **Save.** Do not close the program.

   To define the variables used by the calling program, see "Creating Variables for the Calling Program."

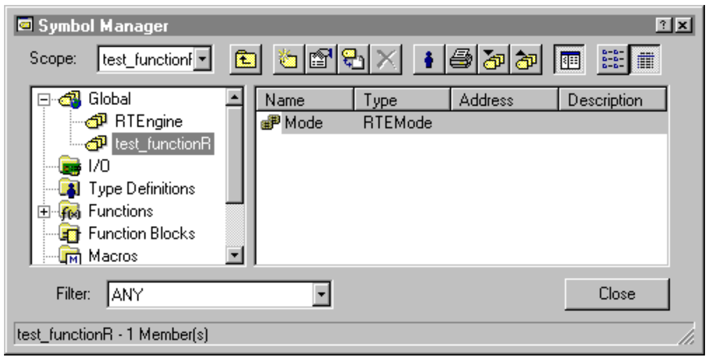
Creating Variables for the Calling Program

This section describes how to define variables for the program that calls the function.

1. 1. If you have closed the calling program, double-click the program name to open it.

   Note that it is not necessary for the program to be open when you define its variables. However, when the program is open, any local variables that you define are associated with the program by default.

2. Click **Symbol Manager** on the **Tools** menu. The Symbol Manager appears.



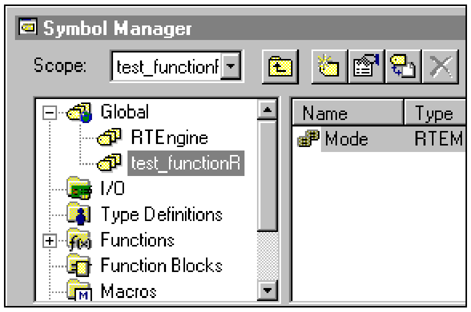
Scope: test_functionR

Name	Type	Address	Description
Mode	RTEMode		

Filter: ANY

test_functionR - 1 Member(s)

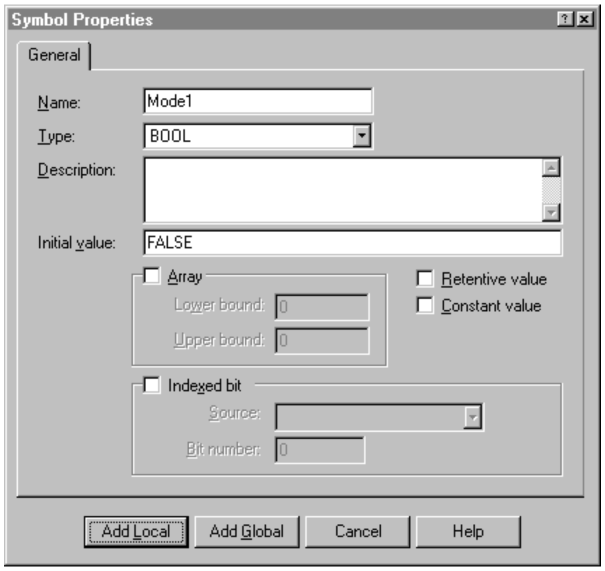
3. Click New on the toolbar.



Scope: test_functionR

Name	Type
Mode	RTEM

The Symbol Properties dialog box appears.



Name: Mode1
Type: BOOL
Description:
Initial value: FALSE
Array: unchecked (Lower bound: 0, Upper bound: 0)
Retentive value: unchecked
Constant value: unchecked
Indexed bit: unchecked (Source: , Bit number: 0)

4. Enter AddResult in the **Name** field.

5. Click **REAL** in the **Type** field.

6. Click **Add Local** to add the new variable name to the Symbol Manager as a local variable.

7. Before closing the Symbol Manager, repeat steps 2-6 to add the remaining variables used by the calling program. All the variables are listed in the table below. Order does not matter for these variables.

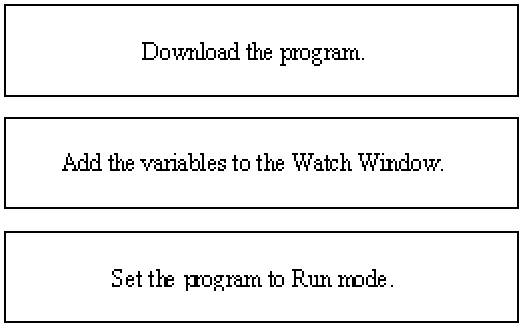
   Continue the example by downloading and running the project. See "Calling and Running the Function."

The variables used by the calling program are listed in the following table.

Name	Data Type
AddResult	REAL
add_in1	REAL
add_in2	REAL

Calling and Running the Function

This section describes how to run the function. The following is the general procedure that you will follow:



Download the program.

Add the variables to the Watch Window.

Set the program to Run mode.

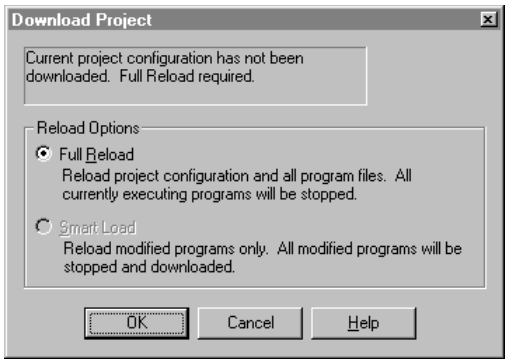
Note that it is necessary to download and run the project because all the code, the calling program and the function, must be loaded to the runtime engine.

Downloading the Project

To download the project:

1. On the **Runtime** menu, click **Connect.** This connects the Development environment to the runtime engine. Ignore this step if you already connected to the runtime engine in order to run any of the other example programs.

2. On the **Runtime** menu, click **Download Project.** The **Download Project** dialog box appears.


Download Project
Current project configuration has not been downloaded. Full Reload required.
Reload Options
Full Reload
Reload project configuration and all program files. All currently executing programs will be stopped.
Smart Load
Reload modified programs only. All modified programs will be stopped and downloaded.
OK
Cancel
Help

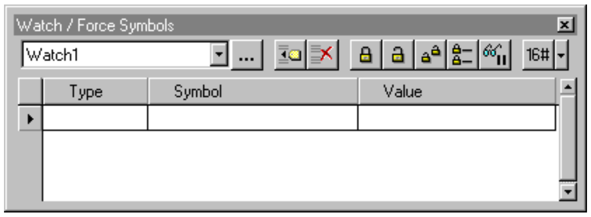
RLL Function Example: Downloading the Project

3. Click **OK** to do a full reload of the runtime engine. The project is downloaded to the runtime engine and all programs are set to the Pause mode.

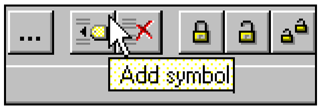
Adding Variables to the Watch Window

To add the program variables to the Watch Window:

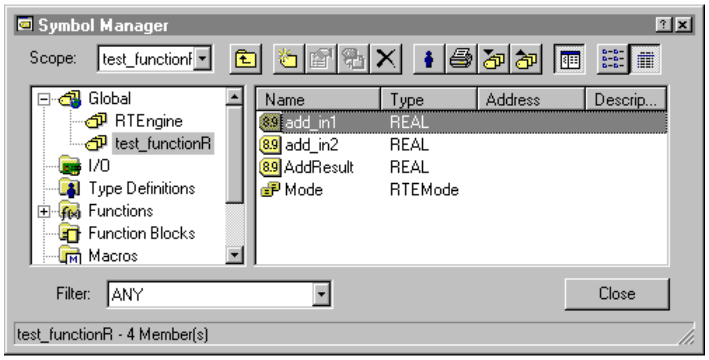
1. If the Watch Window is not visible, on the **View** menu, click **Watch/Force Variables.** The Watch window appears.



2. To add the variables to the Watch window, click **Add Symbol.**



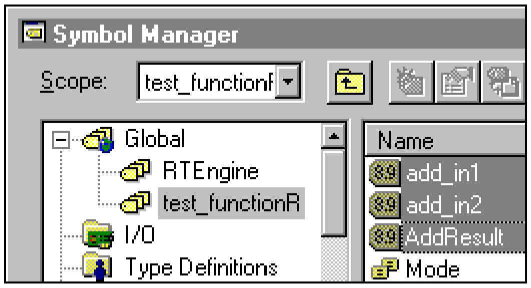
3. The **Symbol Manager** dialog box appears. The variables that you defined are displayed in the Symbol Manager.



Name	Type	Address	Descrip...
add_in1	REAL		
add_in2	REAL		
AddResult	REAL		
Mode	RTEMode		

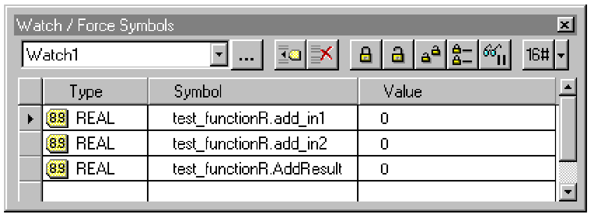
test_functionR - 4 Member(s)

4. Click the following variable names to select them, and click **OK:** add_in1, add_in2, and AddResult. Use the **Ctrl** and **Shift** keys to select multiple items.



RLL Function Example: Selecting Variables for Watch Window

The variables are added to the Watch window.



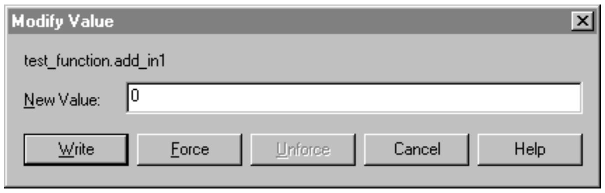
Type	Symbol	Value
REAL	test_functionR.add_in1	0
REAL	test_functionR.add_in2	0
REAL	test_functionR.AddResult	0

Setting the Project to Run Mode

To run the project:

1. On the **Runtime** menu, click **Run Project.**
2. Click **Continue** to run the project. This is the default option, even if you have not run the project before, because the programs were downloaded and then paused.
3. Note the values for add_in1 and add_in2 are both equal zero.

4. Double-click the value of add_in1 in the Watch window. The Modify Value dialog box appears. The default new value to write to the add_in1is zero.



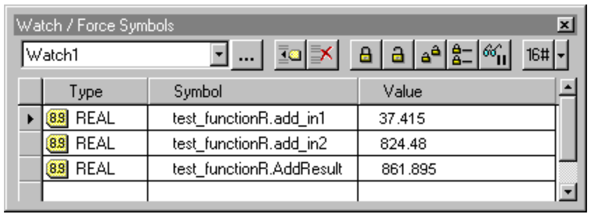
test_function.add_in1

New Value: 0

5. Enter 37.415 and click **Write** to enter the value in add_in1 to be added.

6. Enter 824.48 for the value for add_in2. The RLL program test_functionR, which is continually calling calc_addR, stores the result of the addition to AddResult.

7. Note the result of the addition in AddResult.



Type	Symbol	Value
REAL	test_functionR.add_in1	37.415
REAL	test_functionR.add_in2	824.48
REAL	test_functionR.AddResult	861.895

RLL Function Example: Monitoring the Variables

A P P E N D I X B

SFC Example Program

This appendix presents examples for how to design an STL program, a function, and a function block.

Contents

Developing an SFC Program

This section describes how to create a new SFC program.

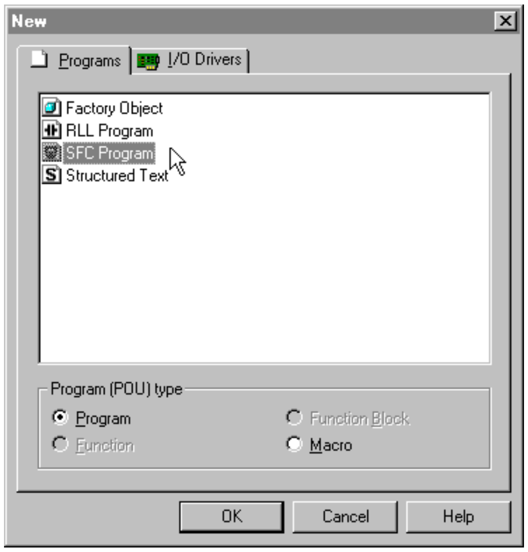
Note The figures in this section are based on a new project with no other programs. If you have already created an RLL program, for example, you may observe some minor differences in the dialog boxes.

Creating a New SFC Program

To begin developing an SFC program:

1. On the **File** menu, click **New**.

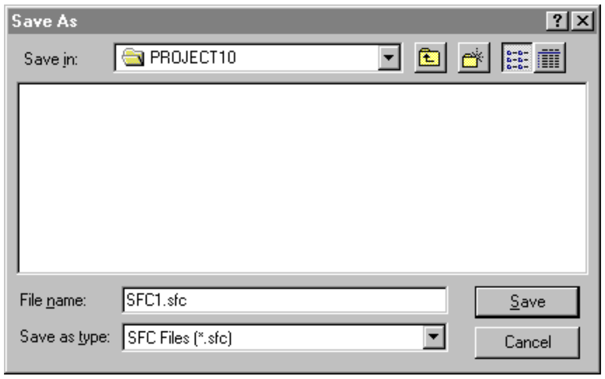
   The **New** dialog box appears.



   SFC Example: New Dialog Box

2. Click SFC Program.
3. Be sure that Program is the selected Program Type. Then click OK.

   The **Save As** dialog box appears.



File name: SFC1.sfc

Save as type: SFC Files (*.sfc)

4. Choose the default or a new name (up to 31 characters) for the program and click **Save.** The default extension .sfc is appended when the file is saved in your project.

The SFC editor displays a new SFC file with a Start Step and an End. To begin editing the program, start by adding a Step. See "Adding a Step."

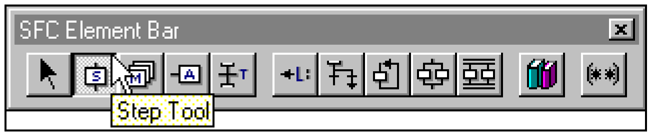
For a detailed description of the SFC editor and toolbar items, see the "Using the SFC Editor" chapter.

Adding a Step

The SFC Step represents a condition in which the operation of the factory process is defined by the code contained within the Step. An SFC can consist of one Step or many Steps of code.

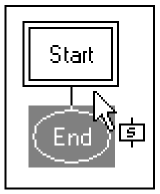
To add a Step to the program:

1. Click **Step Tool** on the SFC toolbar.

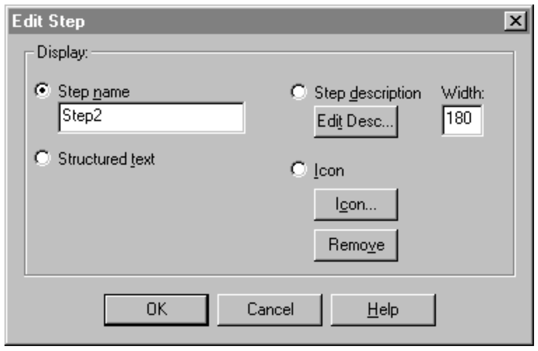


Note that the SFC toolbar may be docked and in a vertical orientation.

2. Move the cursor to the location in the program between the Start and End Steps.



3. Click the left mouse button. The Edit Step dialog box appears.

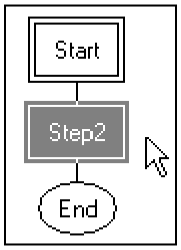


Display:
- (●) Step name: Step2
- ( ) Structured text
- ( ) Step description
- ( ) Icon
- Width: 180

SFC Example: Edit Step Dialog Box

4. Click **OK** to accept the default settings.

The new Step appears in the program at the location you specified.



Entering Code for the Step

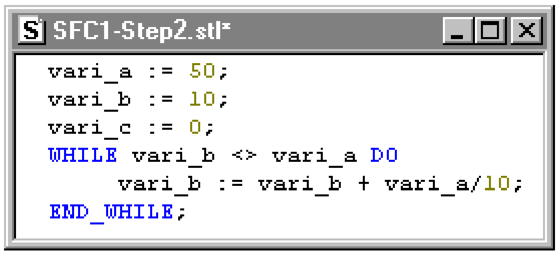
Note You can copy the code from the online manual, paste it directly into the Step, and avoid typing it in manually.

1. Double-click the new Step. A Structured Text editor window appears.

2. Enter the following lines of code:

```
vari_a := 50;
vari_b := 10;
vari_c := 0;
WHILE vari_b <> vari_a DO
   vari_b := vari_b + vari_a/10;
END_WHILE;
```

The following figure shows the contents of the Step after the code is entered:



```
vari_a := 50;
vari_b := 10;
vari_c := 0;
WHILE vari_b <> vari_a DO
     vari_b := vari_b + vari_a/10;
END_WHILE;
```

SFC Example: Code Example 1

3. Close the Step and save when prompted. Do not close the SFC program.

For a description of the operation of the WHILE DO statement, see the "Structured Text Language" chapter.

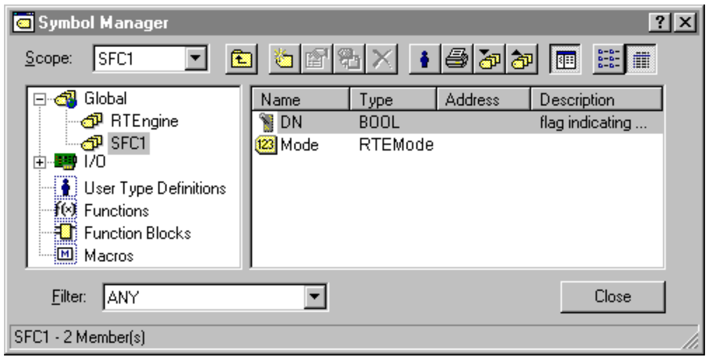
Creating Variables for the SFC Program

This section describes how to define variables for the program.

1. If you have closed the SFC program, double-click the program name to open it.

   It is not necessary for the program to be open when you define its variables. However, when the program is open, any local variables that you define are associated with the program by default.

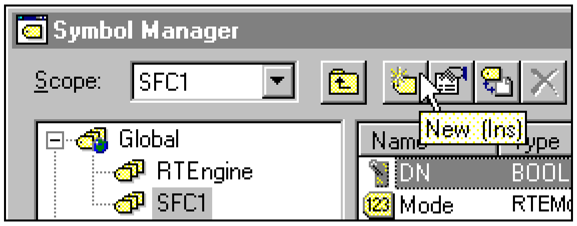
2. Click **Symbol Manager** on the **Tools** menu. The Symbol Manager appears.



Name	Type	Address	Description
DN	BOOL		flag indicating ...
Mode	RTEMode		

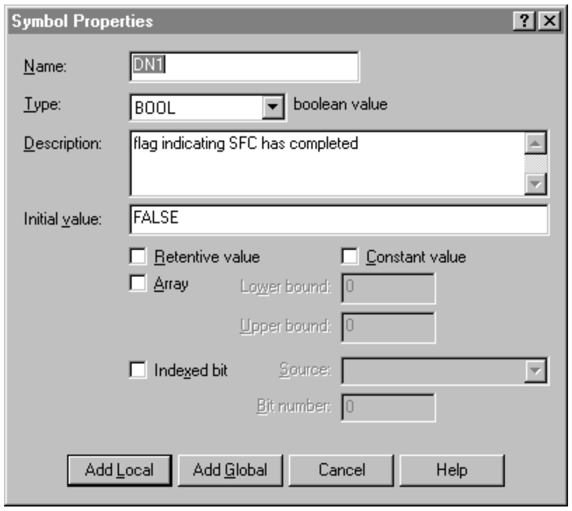
SFC1 - 2 Member(s)

3. Click **New** on the toolbar.



SFC Example: Adding a Symbol

The **Symbol Properties** dialog box appears.



Name: DN1

Type: BOOL boolean value

Description: flag indicating SFC has completed

Initial value: FALSE

Lower bound: 0

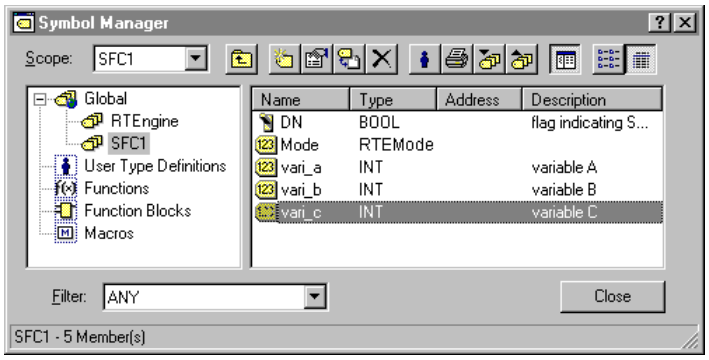
Upper bound: 0

Bit number: 0

4. Enter vari_a in the **Name** field.
5. Click **INT** in the **Type** field. To see a list of all the data types, click the display tool to the right of the field.
6. Give the symbol a meaningful description.
7. Click **Add Local** to add the new variable name to the Symbol Manager as a local variable. This allocates an internal memory location to represent the new variable.
8. Before closing the Symbol Manager, repeat steps 2-5 to add integers vari_b and vari_c to the Symbol Manager. Define both variables as INT data types.
9. Click **Close** to close the Symbol Manager.

For more information about defining variables, see "Creating a Variable" in the "Defining Variables" chapter.

The following figure shows the contents of the Symbol Manager after all three variables have been defined. The integer variable called Mode is a system variable that contains the value of the program mode (Run, Pause, Stop, etc.).



Name	Type	Address	Description
DN	BOOL		flag indicating S...
Mode	RTEMode		
vari_a	INT		variable A
vari_b	INT		variable B
vari_c	INT		variable C

SFC Example: New Symbols

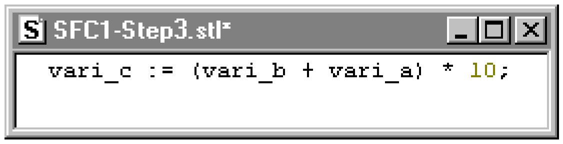
Adding a Second Step

Note You can copy the code from the online manual, paste it directly into the Step, and avoid typing it in manually.

Using the procedure described on the preceding pages, add a second Step to the SFC. Use the following line of code:

```
vari_c := (vari_b + vari_a) * 10;
```

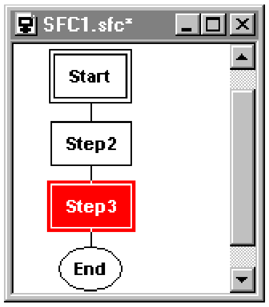
The following figure shows the contents of the Step after the code is entered:



```
vari_c := (vari_b + vari_a) * 10;
```

SFC Example: Code Example 2

The following figure shows the SFC after both Steps have been entered:



SFC Example: Two New Steps

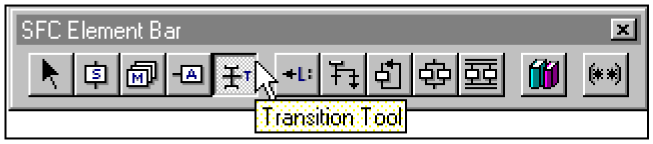
The asterisk by the program name in the window title bar indicates that the program has not been saved; or when the SFC is executing on the runtime engine, that this is not the same version because editing changes have been made.

Adding a Transition

The Transition represents the condition that allows program flow to pass from one or more Steps preceding the Transition to one or more Steps following the Transition. Two types of Transitions are available, the Boolean Transition and the RLL Transition. This example uses Boolean for the Transition logic. In either case, Transition logic always resolves to either a TRUE or FALSE value.

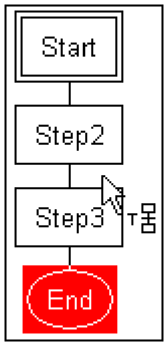
To add a Boolean Transition to the program:

1. Click **Edit** in the menu bar. If **Boolean Transitions** does not have a check by it, click **Boolean Transitions** to select it.
2. Click **Transition Tool** on the SFC toolbar.



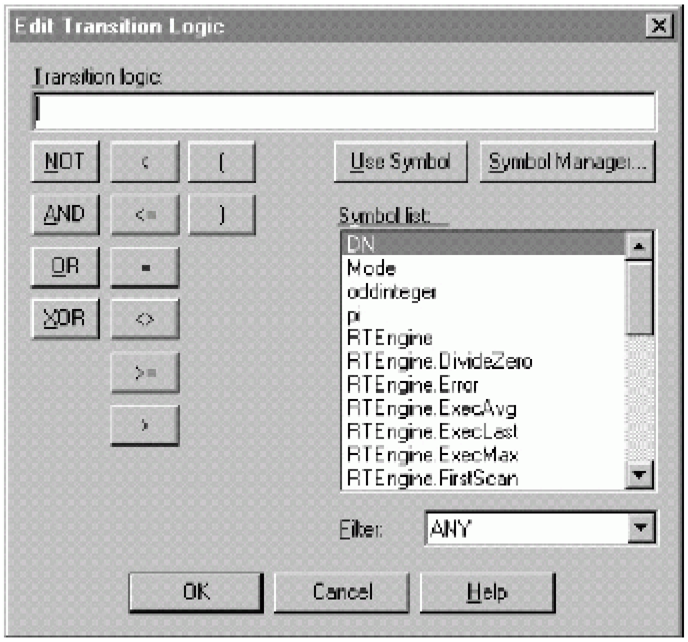
The cursor changes into the Transition cursor.

3. Move the cursor to the location in the program between the Steps.



4. Click the left mouse button. The new Transition appears in the program.

5. Double-click the new Transition. The **Edit Transition Logic** dialog box appears.

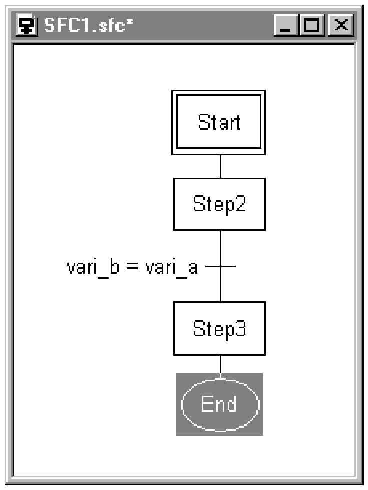


SFC Example: Transition Code

6. Enter the following code in the **Transition Logic** field and click **OK:**

```
vari_b = vari_a
```

The following figure shows the completed SFC quickstart example after the Transition and Steps have been added.



SFC Example: New Transition

On the **File** menu, click **Save** to save your work. To validate, download, and run the program, see "Running the SFC Program." Note that InControl automatically saves any edit changes when you execute a Validate, Download, or Run command.

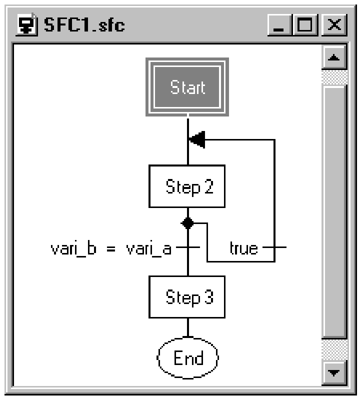
Alternative Looping

You can use a Loop, instead of a WHILE statement to handle the loopback to Step1. The SFC requires the following changes:

- When you define the variables in the Symbol Manager, assign them initial values. Set vari_a = 50; vari_b = 10; and vari_c = 0. When you enter the code for Step1, do not include the assignment statements for these variables.
- Remove the WHILE statement from Step1.

  The alternative code for Step1 consists of the following line:

  ```
  vari_b := vari_b + vari_a/10;
  ```
- Replace the Transition with a Loop. The Loop construction is shown in the figure below. Double-click the two Loop Transitions to edit them. The conditions for the Transitions are shown in the following figure.



SFC Example: Alternative Looping

Note This alternative method takes more scans to execute. For more information see "SFC Execution" in the "InControl System Administration" chapter.

Running the SFC Program

This section describes how to run the SFC program. Because of its simple design, this program is best observed if you single scan it instead of allowing it to run to completion. This allows you to observe the variables as their values change. The following is the general procedure that you will follow:

Download the program.

Add the variables to the Watch Window.

Single scan the program.

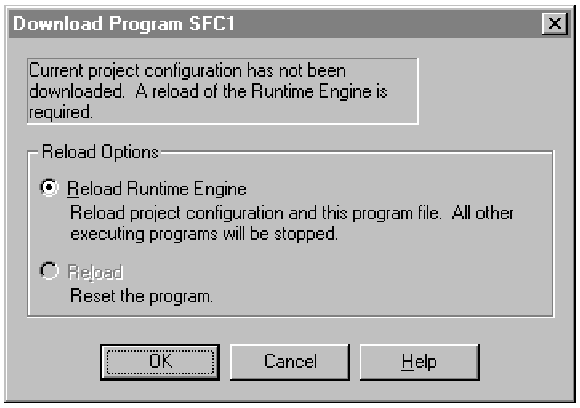
Downloading the SFC Program

To download the SFC program:

1. On the **Runtime** menu, click **Connect.** This connects the Development environment to the runtime engine.

   Ignore this step if you have already connected to the runtime engine in order to run the RLL example program.

2. On the **Runtime** menu, click **Download Program**.The **Download Program** dialog box appears.



Download Program SFC1

Current project configuration has not been downloaded. A reload of the Runtime Engine is required.

Reload Options

[✓ Reload Runtime Engine]
Reload project configuration and this program file. All other executing programs will be stopped.

Reload
Reset the program.

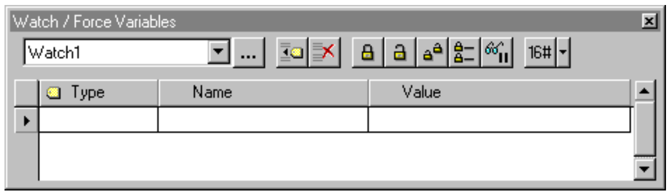
SFC Example: Download Program Dialog Box

3. Click **OK** to reload the runtime engine. The SFC is downloaded to the runtime engine and set to the Pause mode.

Adding Variables to the Watch Window

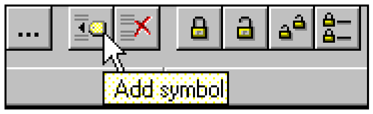
To add the SFC program variables to the Watch Window:

1. If the Watch Window is not visible, on the **View** menu, click **Watch/Force Variables.** The Watch window appears.

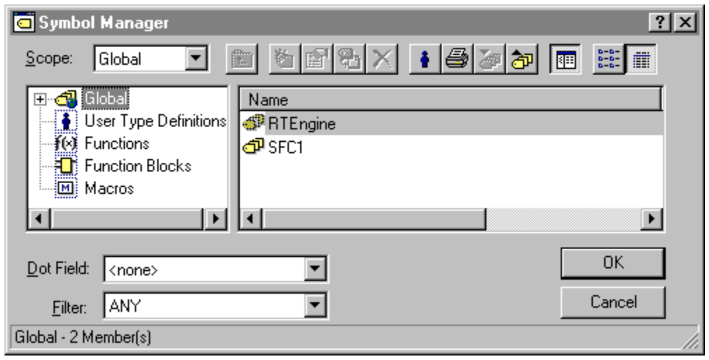


SFC Example: Watch Window 1

2. To add the variables to the Watch window, click **Add Symbol.**

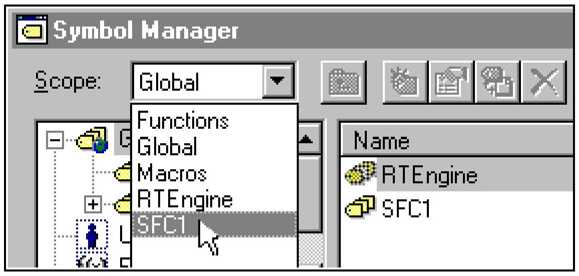


The **Symbol Manager** dialog box appears.



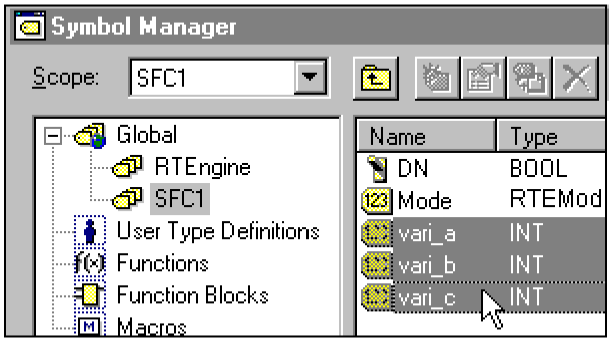
SFC Example: Symbol Manager 1

3. Since the SFC variables are local, click the display tool for the **Scope** field and select your SFC program.



SFC Example:Selecting Scope

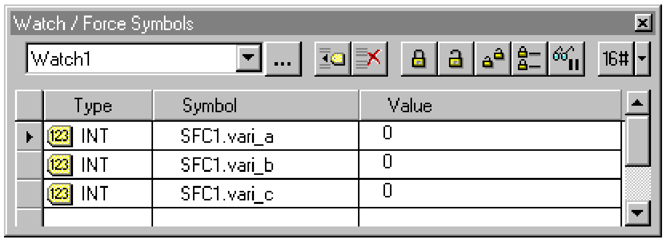
The variables that you defined are displayed in the Symbol Manager. The two variables (DN and Mode) are SFC system variables.



Name	Type
DN	BOOL
Mode	RTEMod
vari_a	INT
vari_b	INT
vari_c	INT

SFC Example: Selecting Variables

4. Click the variable names to select them, and click **OK.** Use the **Ctrl** and **Shift** keys to select multiple items. The variables are added to the Watch window.



Type	Symbol	Value
INT	SFC1.vari_a	0
INT	SFC1.vari_b	0
INT	SFC1.vari_c	0

SFC Example: Symbol Manager 2

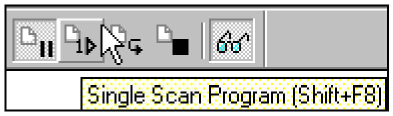
Single Scanning the SFC Program

Because of its simple design, this program is best observed if you single scan it instead of allowing it to run to completion. This allows you to observe the variables as their values change.

To single scan the SFC program:

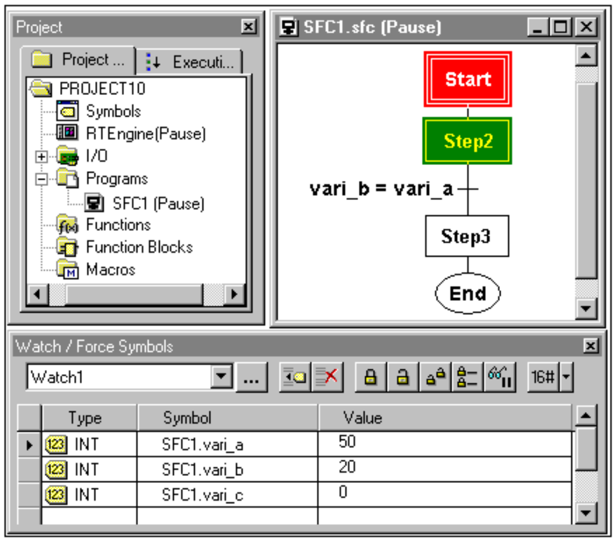
1. If you have not already connected to the runtime engine, on the **Runtime** menu, click **Connect.**

2. On the **Runtime** menu, click **Single Scan Program** to cause the SFC to execute for one scan.



You can also press **Shift F8** or click the **Single Scan Program** tool.

3. As the SFC executes during a scan, observe the values of the variables change in the Watch Window. The runtime highlighting indicates which program element is being executed.



Type	Symbol	Value
INT	SFC1.vari_a	50
INT	SFC1.vari_b	20
INT	SFC1.vari_c	0

SFC Example: Single-Scanning the Program

Note If the program elements are not highlighted, on the **View** menu, click **Runtime Highlighting.**

4. 4. Continue to single scan the program. When the variable vari_c equals 1000 and program flow reaches the End Step, the SFC finishes its execution and enters a Complete mode.

5. To execute the SFC again, download the SFC again and resume single stepping it. When the **Download Program** dialog box appears, you can click **Reload**, which avoids compiling the program again.

   If you chose to use a Loop in the program, as described in "Alternative Looping" click **Reload Runtime Engine** when the **Download Program** dialog box appears. Otherwise, the variables are not reinitialized.

Add other SFC program elements to the SFC program to become familiar with the rest of the SFC programming editor.

A P P E N D I X C

STL Example Program

This appendix presents examples for how to design an STL program, a function, and a function block.

Contents

Developing a Structured Text Program

This section describes how to create a new Structured Text (STL) program.

Note The figures in this section are based on a new project with no other programs. If you have already created an RLL program, for example, you may observe some minor differences in the dialog boxes.

Creating a New STL Program

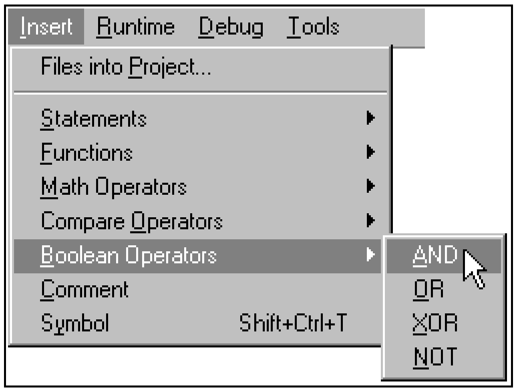
To begin developing a STL program:

1. On the **File** menu, click **New.** The **New** dialog box appears.
2. Click **Structured Text Program.**
3. Be sure that Program is the selected Program Type. Then click OK. The **Save As** dialog box appears.
4. Choose the default or a new name (up to 31 characters) for the program and click **Save.** The default extension .stl is appended when the file is saved in your project.

   The STL editor displays a new Structured Text file ready for editing. To begin editing the program, see "Entering STL Code."

Entering STL Code

You can type Structured Text code directly into the editor window or by making selections from the **Insert** menu, which is shown in the following figure.



STL Example: Selecting Functions

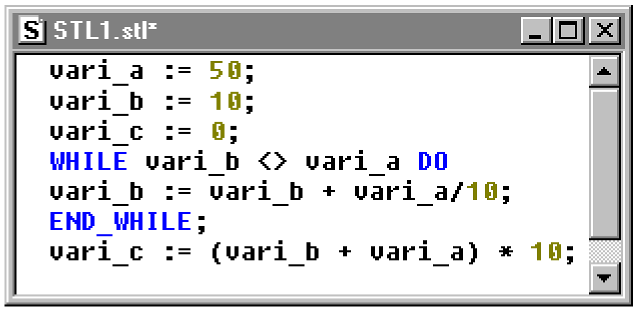
To enter the example code:

Note You can copy the code from the online manual, paste it directly into the editor window, and avoid typing it in manually. If you created the SFC example, you can copy the code from the SFC Steps.

1. Type the following lines of code:

```
vari_a := 50;
vari_b := 10;
vari_c := 0;
WHILE vari_b<>vari_a DO
   vari_b:=vari_b+vari_a/10;
END_WHILE;
vari_c:=(vari_b+vari_a)*10;
```

The following figure shows the contents of the Structured Text editor after the code is entered:



```
vari_a := 50;
vari_b := 10;
vari_c := 0;
WHILE vari_b <> vari_a DO
vari_b := vari_b + vari_a/10;
END_WHILE;
vari_c := (vari_b + vari_a) * 10;
```

STL Example: Entering Code

The asterisk by the program name in the window title bar indicates that the program has not been saved; or when the program is executing on the runtime engine, that this is not the same version because editing changes have been made.

2. On the **File** menu, click **Save.** Do not close the program.

Creating Variables for the STL Program

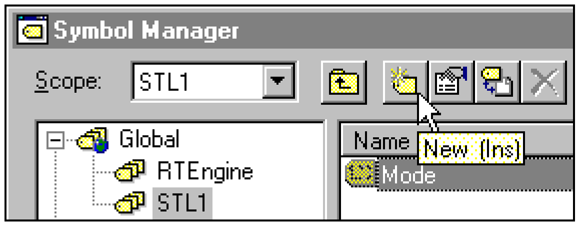
This section describes how to define variables for the program.

1. If you have closed the program, double-click the program name to open it.

   Note that it is not necessary for the program to be open when you define its variables. However, when the program is open, any local variables that you define are associated with the program by default.

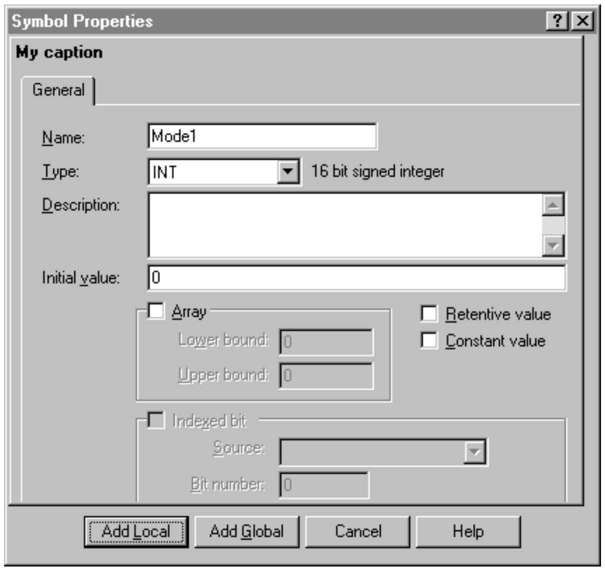
2. Click **Symbol Manager** on the **Tools** menu. The Symbol Manager appears.

3. Click **New** on the toolbar.



STL Example: Adding a Symbol

The **Symbol Properties** dialog box appears.


Symbol Properties

My caption

General

Name: Mode1

Type: INT 16 bit signed integer

Description:

Initial value: 0

☐ Array Lower bound: 0 Upper bound: 0

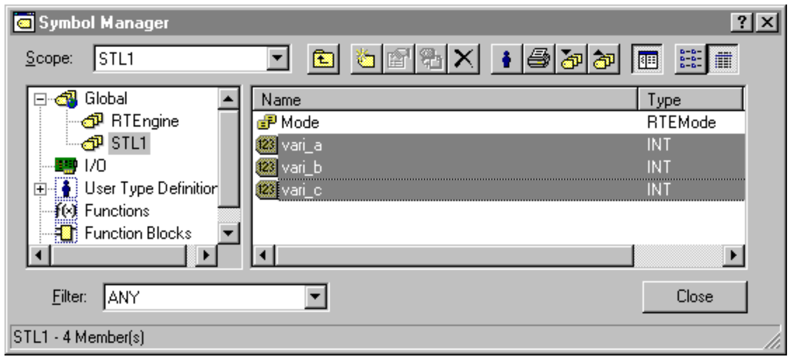
☐ Retentive value

☐ Constant value

☐ Indexed bit Source: Bit number: 0

4. Enter vari_a in the **Name** field.
5. Click **INT** in the **Type** field. To see a list of all the data types, click the display tool to the right of the field.
6. Click **Add Local** to add the new variable name to the Symbol Manager as a local variable. This allocates an internal memory location to represent the new variable.
7. Before closing the Symbol Manager, repeat steps 2-5 to add integers vari_b and vari_c to the Symbol Manager. Define both variables as INT data types.
8. Click **Close** to close the Symbol Manager.

The following figure shows the contents of the Symbol Manager after all three variables have been defined. The integer variable called Mode is a system variable that contains the value of the program mode (Run, Pause, Stop, etc.).


Symbol Manager

Scope: STL1

Name	Type
Mode	RTEMode
vari_a	INT
vari_b	INT
vari_c	INT

Filter: ANY

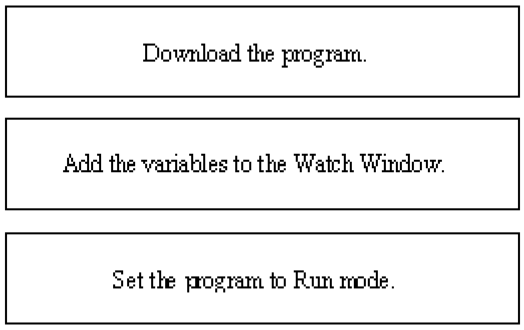
STL1 - 4 Member(s)

STL Example: New Symbols

To validate, download, and run the program, see "Running the STL Program." Note that InControl automatically saves any edit changes when you execute a Validate, Download, or Run command.

Running the STL Program

This section describes how to run the Structured Text program. The following is the general procedure that you will follow:



Download the program.

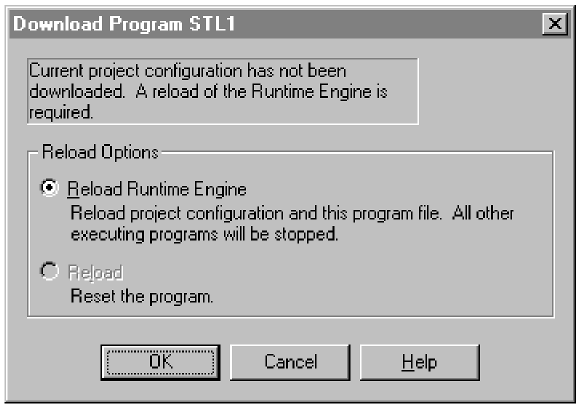
Add the variables to the Watch Window.

Set the program to Run mode.

Downloading the Structured Text Program

To download the Structured Text program:

1. On the **Runtime** menu, click **Connect.** This connects the Development environment to the runtime engine. Ignore this step if you already connected to the runtime engine in order to run the RLL or SFC programs.
2. On the **Runtime** menu, click **Download Program.** The **Download Program** dialog box appears.



Download Program STL1

Current project configuration has not been downloaded. A reload of the Runtime Engine is required.

Reload Options

[✓ Reload Runtime Engine]
Reload project configuration and this program file. All other executing programs will be stopped.

Reload
Reset the program.

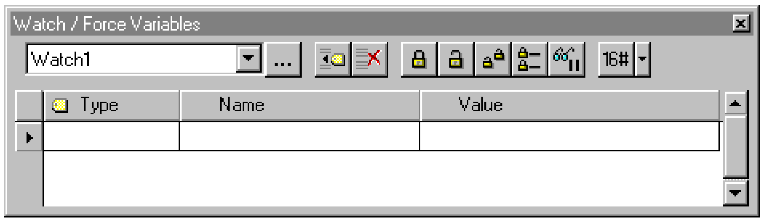
STL Example: Download Program Dialog Box

3. Click **OK** to reload the runtime engine. The program is downloaded to the runtime engine and set to the Pause mode.

Adding Variables to the Watch Window

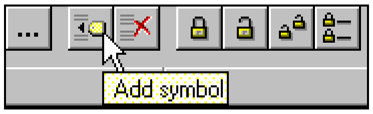
To add the program variables to the Watch Window:

1. If the Watch Window is not visible, on the **View** menu, click **Watch/Force Variables.** The Watch window appears.

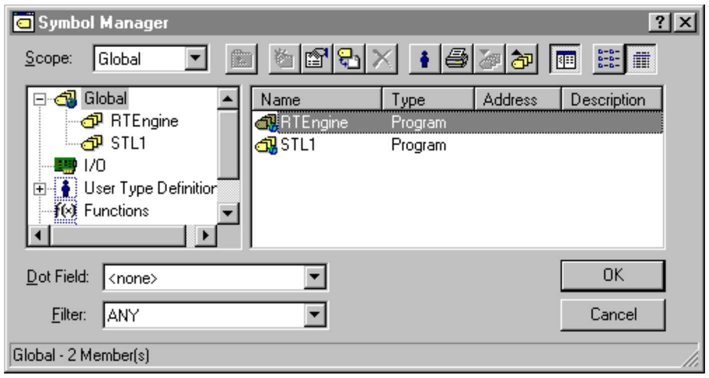


STL Example: Watch Window 1

2. To add the variables to the Watch window, click **Add Symbol.**



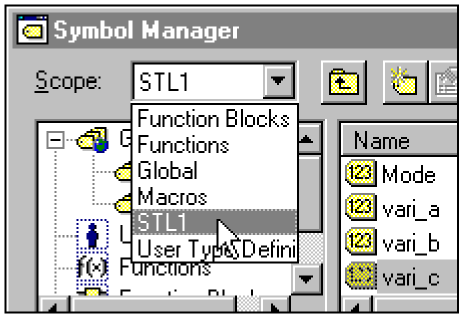
The Symbol Manager dialog box appears.



Name	Type	Address	Description
RTEngine	Program		
STL1	Program		

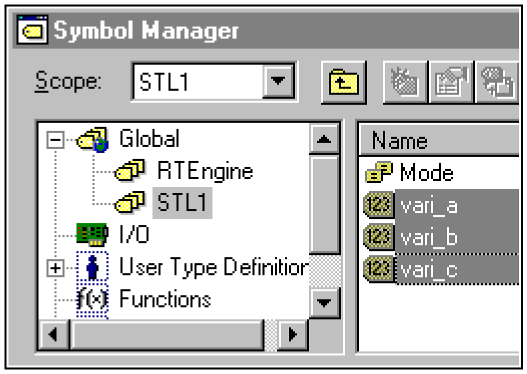
STL Example: Symbol Manager 1

3. Since the variables are local, click the display tool for the **Scope** field and select **STL1.**



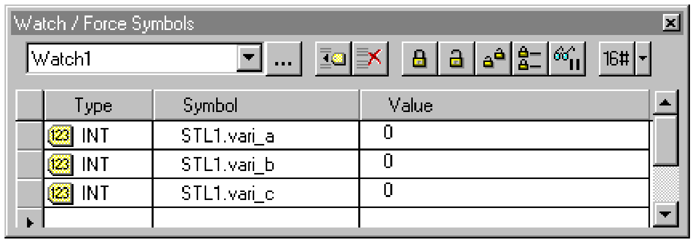
STL Example: Selecting Scope

The Structured Text system variable (Mode) and the variables that you defined are displayed in the Symbol Manager.



STL Example: Selecting Variables

4. Click the variable names to select them, and click **OK.** Use the **Ctrl** and **Shift** keys to select multiple items. The variables are added to the Watch window.



Type	Symbol	Value
INT	STL1.vari_a	0
INT	STL1.vari_b	0
INT	STL1.vari_c	0

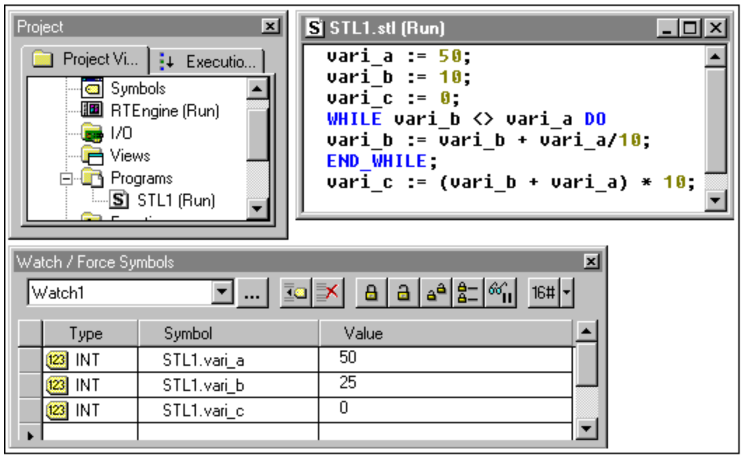
STL Example: Symbol Manager 2

Setting the Program to Run Mode

To run the program:

1. On the **Runtime** menu, click **Run Program.**
2. Click **Continue** to run the program. This is the default option, even if you have not run the program before, because the program was downloaded and then paused.

3. As the program runs, observe the values of the variables change in the Watch Window.



```
vari_a := 50;
vari_b := 10;
vari_c := 0;
WHILE vari_b <> vari_a DO
vari_b := vari_b + vari_a/10;
END_WHILE;
vari_c := (vari_b + vari_a) * 10;
```

Type	Symbol	Value
INT	STL1.vari_a	50
INT	STL1.vari_b	25
INT	STL1.vari_c	0

STL Example: Running the Program

Add other Structured Text program elements to the program to become familiar with the rest of the Structured Text programming editor.

Developing a Function Block

This section describes how to create a new function block. The following is the general procedure that you will follow:

Create a new function block type and enter its code.

Define the parameters for the function block type.

Add an instance of the function block to the project.

Create a new Structured Text program and enter code that calls the function block for execution.

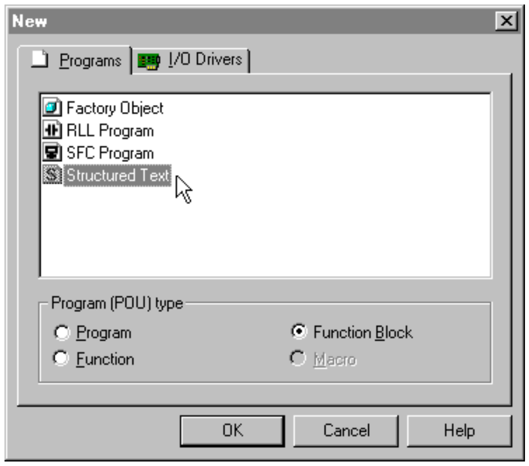
Note The figures in this section are based on a new project with no other programs. If you have already created an RLL program, for example, you may observe some minor differences in the dialog boxes.

Creating a New Function Block Type

To begin developing a function block type:

1. On the File menu, click New.

   The New dialog box appears.



Program (POU) type: Program ( ) Function ( ) [✓ Function Block] Macro ( )

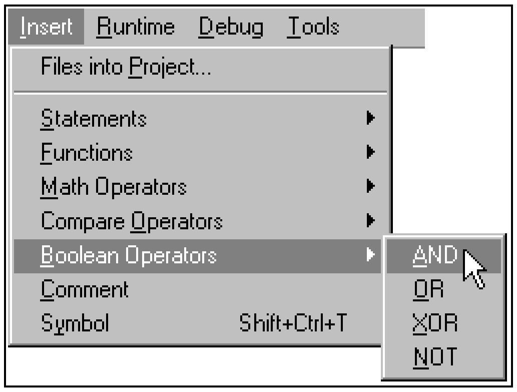
Function Block Example: New Dialog Box

2. Click **Structured Text** Program.
3. Click **Function Block** to select the Program Type. Then click OK. The **Save As** dialog box appears.
4. Enter a name for the function block (up to 31 characters) and click Save.This example uses fbedge.stl for the name. The default extension .stl is appended when the file is saved in your project.

   The Structured Text editor displays a new Structured Text file ready for editing. To begin editing the function block, see "Entering Function Block Code."

Entering Function Block Code

You can type Structured Text code directly into the editor window or by making selections from the **Insert** menu, which is shown in the following figure.



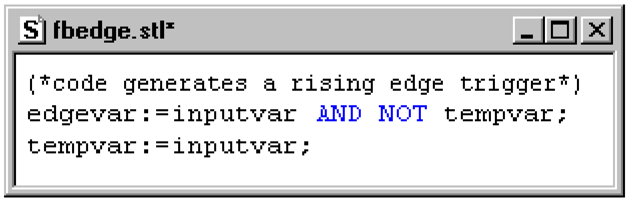
Function Block Example: Selecting Functions

To enter the example code:

Note You can copy the code from the online manual, paste it directly into the editor window, and avoid typing it in manually.

1. Type the following lines of code:
   ```
   (*code generates a rising edge trigger*)
   edgevar:=inputvar AND NOT tempvar;
   tempvar:=inputvar;
   ```

   The following figure shows the contents of the Structured Text editor after the code is entered:

   ```
   (*code generates a rising edge trigger*)
   edgevar:=inputvar AND NOT tempvar;
   tempvar:=inputvar;
   ```

   Function Block Example: Entering Code

   The asterisk by the program name in the window title bar indicates that the program has not been saved; or when the program is executing on the runtime engine, that this is not the same version because editing changes have been made.

2. On the **File** menu, click **Save.**

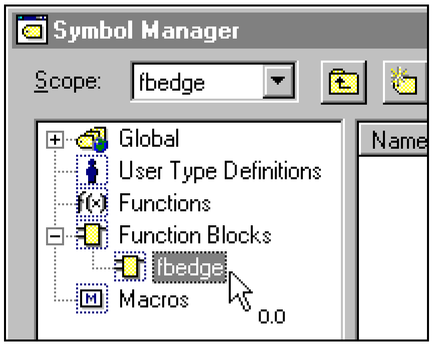
   To define the parameters used by the function block, see "Creating Function Block Parameters."

Creating Function Block Parameters

You define the input and output parameters and variables for a function block type in the Symbol Manager. Function block local variables are local to the function block instance and cannot be referenced elsewhere in the project. Within the function block instance, input parameters are read only. Output parameters must be assigned values through an assignment statement.

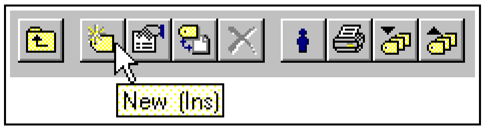
To define a parameter or variable for a function block type:

1. On the **Tools** menu, click **Symbol Manager.**
2. Click the function block type to select it as shown in the following figure.

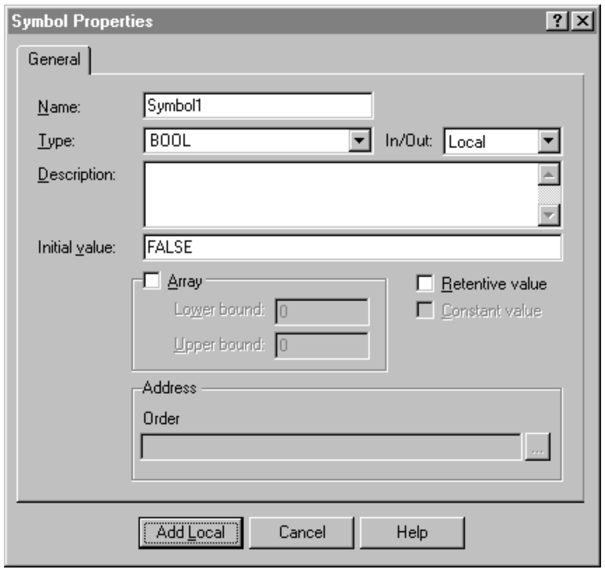


Function Block Example: Selecting the Function Block

3. Click New on the Symbol Manager toolbar.



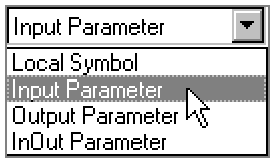
The **Symbol Properties** dialog box appears.



Name: Symbol1
Type: BOOL
In/Out: Local
Description:
Initial value: FALSE
Lower bound: 0
Upper bound: 0

4. Enter inputvar for the name of the first parameter into the **Name** field.

5. Select BOOL as the data type in the **Type** field.
6. Enter the optional description into the **Description** field. This example uses the following text:
   Variable to detect if input variable transitions from FALSE to TRUE.
7. Choose Input Parameter in the **In/Out** field, as shown below.

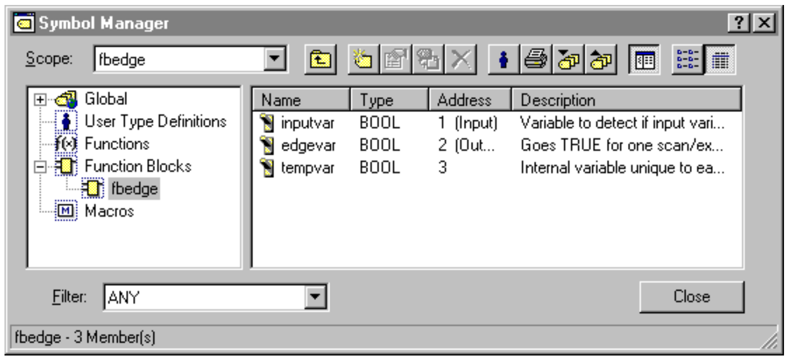


8. Click **Add Local** to complete the definition for this parameter.
9. Before closing the Symbol Manager, repeat steps 3-8 to add the output parameter and local variable used by the example function block. All three are listed in the table below. Enter them in the order shown.
10. Click **Close** to close the Symbol Manager.

The Structured Text toolbar options are described in the following table.

Name	Data Type	In/Out	Description
inputvar	BOOL	Input	Variable to detect if input variable transitions from FALSE to TRUE.
edgevar	BOOL	Output	Goes TRUE for one scan/execution when the input variable changes from FALSE to TRUE.
tempvar	BOOL	Local Symbol	Internal variable unique to each instance of the function block to help detect the input's transition from FALSE to TRUE.

The following figure shows the contents of the Symbol Manager after the two parameters and the local variable have been defined. The order of the parameters in the Symbol Manager is indicated in the **Address** field.



Scope: fbedge

Name	Type	Address	Description
inputvar	BOOL	1 (Input)	Variable to detect if input vari...
edgevar	BOOL	2 (Out...	Goes TRUE for one scan/ex...
tempvar	BOOL	3	Internal variable unique to ea...

Filter: ANY

fbedge - 3 Member(s)

Function Block Example: Parameters and Variables

If you enter parameters out of order, you can change their order in the Symbol Manager.

To change the order of parameters:

1. Right-click the parameter.
2. Click **Decrease Address** or **Increase Address** to change the order of the parameter in the list.

Creating the Calling Program

A function block does not run automatically, but rather must be called by a program. This section describes how to create a Structured Text program to call the example function block.

To create a Structured Text Program:

1. Follow the procedure that you used to create a function block, as described in "Creating a New Function Block Type."

   On the **File** menu, click **New.**

   Click **Structured Text Program.**

   Click **Program** to select the Program Type. Then click **OK.**

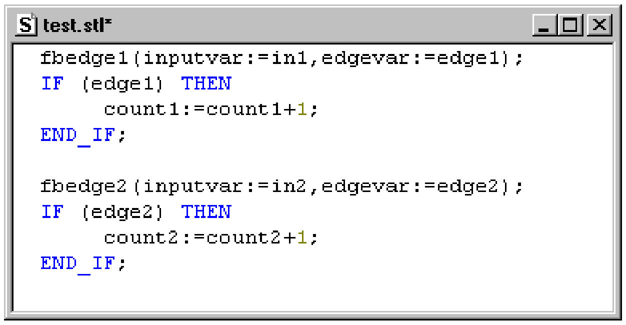
   Enter a name for the program (up to 31 characters) and click **Save**.This example uses test.stl for the name.

   The Structured Text editor displays a new Structured Text file ready for editing.

2. Enter the following code for the program.

```
fbedge1(inputvar:=in1,edgevar:=edge1);
IF (edge1) THEN
    count1:=count1+1;
END_IF;
fbedge2(inputvar:=in2,edgevar:=edge2);
IF (edge2) THEN
    count2:=count2+1;
END_IF;
```

   The following figure shows the contents of the Structured Text editor after the code is entered:



test.stl*

```
fbedge1(inputvar:=in1,edgevar:=edge1);
IF (edge1) THEN
     count1:=count1+1;
END_IF;

fbedge2(inputvar:=in2,edgevar:=edge2);
IF (edge2) THEN
     count2:=count2+1;
END_IF;
```

   Function Block Example: Code for Calling Program

3. On the **File** menu, click **Save.** Do not close the program.

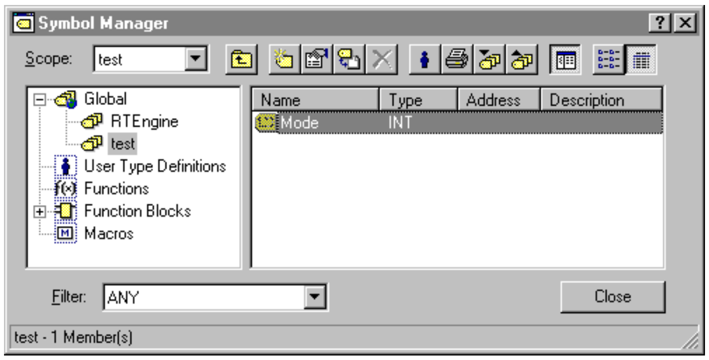
Creating Variables for the Calling Program

This section describes how to define variables for the program that calls the function block.

1. If you have closed the calling program, double-click the program name to open it.

   Note that it is not necessary for the program to be open when you define its variables. However, when the program is open, any local variables that you define are associated with the program by default.

2. Click **Symbol Manager** on the **Tools** menu. The Symbol Manager appears



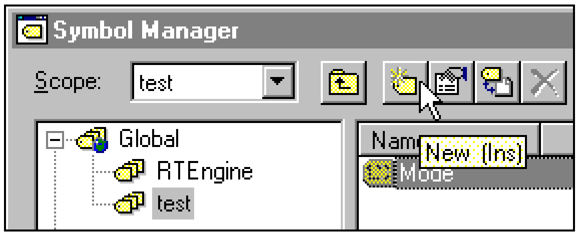
Scope: test

Name	Type	Address	Description
Mode	INT		

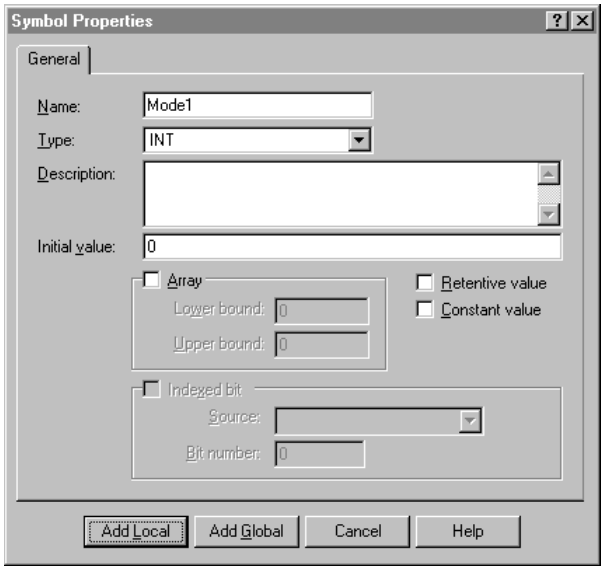
Filter: ANY

test - 1 Member(s)

3. Click **New** on the toolbar.



The **Symbol Properties** dialog box appears.



Name: Mode1
Type: INT
Description:
Initial value: 0

4. Enter edge1 in the **Name** field.
5. Click **BOOL** in the **Type** field.
6. Click **Add Local** to add the new variable name to the Symbol Manager as a local variable.
7. Before closing the Symbol Manager, repeat steps 2-6 to add the remaining variables used by the calling program. All the variables are listed in the table below. Order does not matter for these variables.
8. Do not close the Symbol Manager. Continue the example by adding two instances of the function block. See "Creating the Function Block Instances."

The variables used by the calling program are listed in the following table.

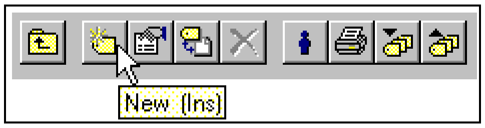
Name	Data Type
edge1	BOOL
edge2	BOOL
in1	BOOL
in2	BOOL
count1	INT
count2	INT

Creating the Function Block Instances

You define an instance of a function block type in the Symbol Manager.

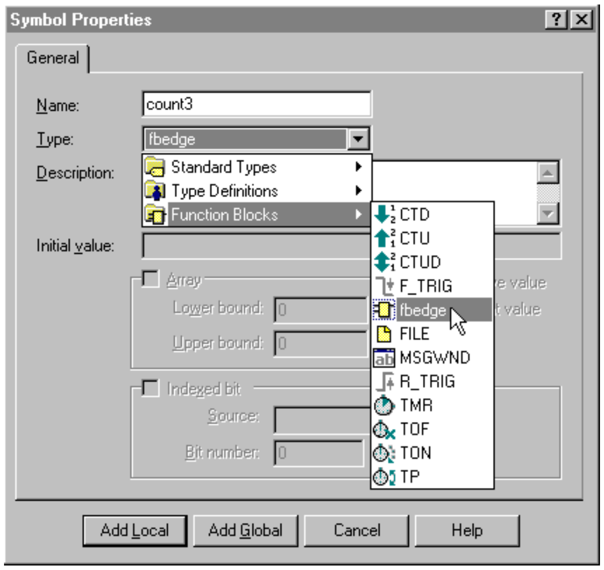
To define an instance of a function block:

1. If you have already closed the Symbol Manager, on the **Tools** menu click **Symbol Manager.** The Symbol Manager appears.
2. Click the name of the calling program test.stl to select the appropriate scoping level.
3. Click **New** on the Symbol Manager toolbar.



The **Symbol Properties** dialog box appears.

4. Enter the name of the first instance into the **Name** field. This example uses fbedge1.
5. Select the function block type fbedge in the Type field.


Symbol Properties

General

Name: count3

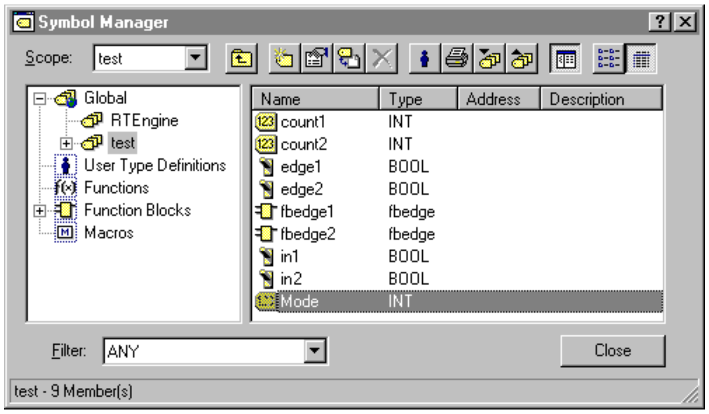
Type: fbedge

Description:

Initial value:

6. Enter an optional description into the **Description** field.
7. Since all the remaining properties are defined by the function block type, click **Add Local** to complete the definition for the instance.
8. Before closing the Symbol Manager, repeat steps 2-7 to add a second instance of the function block to the Symbol Manager. Use fbedge2 for the name.
9. Click **Close** to close the Symbol Manager.

The following figure shows the contents of the Symbol Manager after the variables and function block instances have been defined.



Name	Type	Address	Description
count1	INT		
count2	INT		
edge1	BOOL		
edge2	BOOL		
fbedge1	fbedge		
fbedge2	fbedge		
in1	BOOL		
in2	BOOL		
Mode	INT		

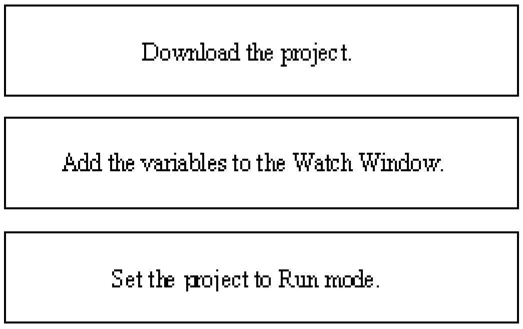
test - 9 Member(s)

Function Block Example: Variables and Function Block Instances

Continue the example by downloading and running the project. See "Calling and Running the Function Block."

Calling and Running the Function Block

This section describes how to run the function block. The following is the general procedure that you will follow:



Download the project.

Add the variables to the Watch Window.

Set the project to Run mode.

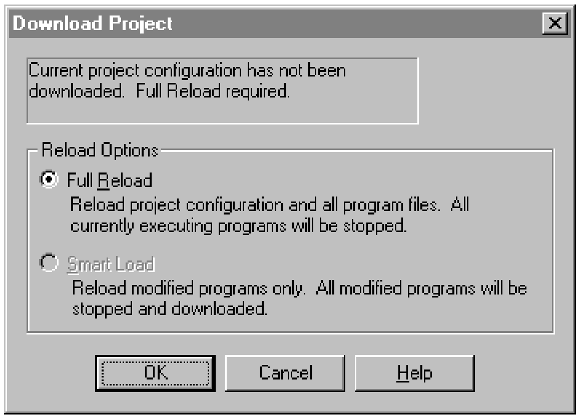
Note that it is necessary to download and run the project because all the code, the calling program and the function block, must be loaded to the runtime engine.

Downloading the Project

To download the project:

1. On the **Runtime** menu, click **Connect.** This connects the Development environment to the runtime engine. Ignore this step if you already connected to the runtime engine in order to run any of the other example programs.

2. On the **Runtime** menu, click **Download Project.** The **Download Project** dialog box appears.


Download Project

Current project configuration has not been downloaded. Full Reload required.

Reload Options

(●) Full Reload
Reload project configuration and all program files. All currently executing programs will be stopped.

( ) Smart Load
Reload modified programs only. All modified programs will be stopped and downloaded.

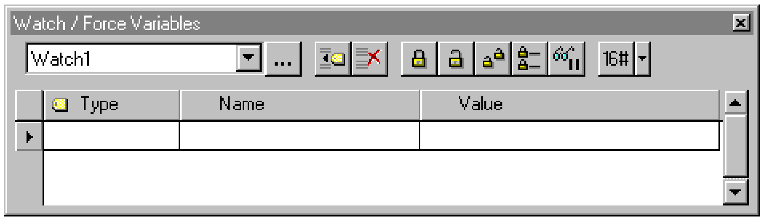
Function Block Example: Downloading the Project

3. Click **OK** to do a full reload of the runtime engine. The project is downloaded to the runtime engine and all programs are set to the Pause mode.

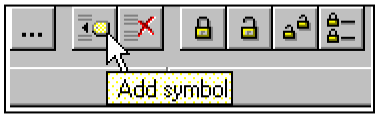
Adding Variables to the Watch Window

To add the program variables to the Watch Window:

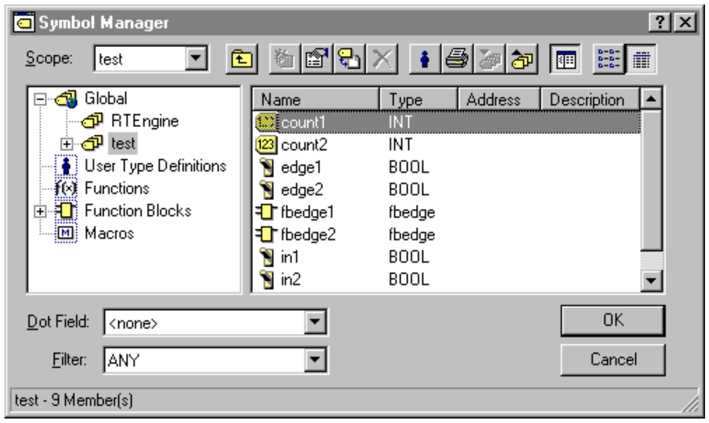
1. If the Watch Window is not visible, on the **View** menu, click **Watch/Force Variables.** The Watch window appears.



2. To add the variables to the Watch window, click **Add Symbol.**



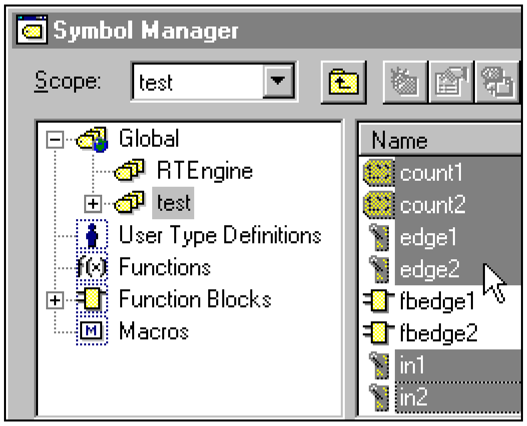
3. The **Symbol Manager** dialog box appears. The variables that you defined are displayed in the Symbol Manager.



Name	Type	Address	Description
count1	INT		
count2	INT		
edge1	BOOL		
edge2	BOOL		
fbedge1	fbedge		
fbedge2	fbedge		
in1	BOOL		
in2	BOOL		

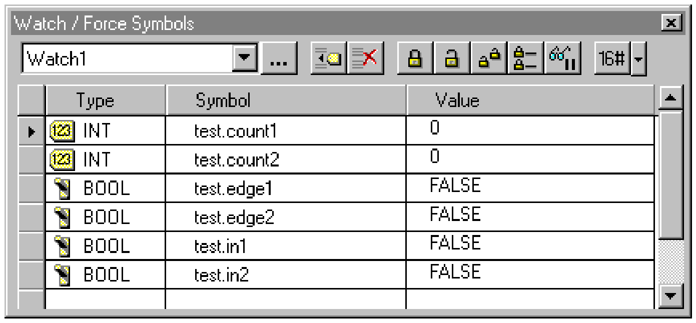
4. Click the following variable names to select them, and click **OK:**

   edge1, edge2, in1, in2, count1, and count2. Use the **Ctrl** and **Shift** keys to select multiple items.



Function Block Example: Selecting Variables for Watch Window

The variables are added to the Watch window.

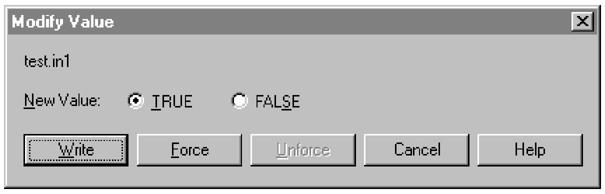


Type	Symbol	Value
INT	test.count1	0
INT	test.count2	0
BOOL	test.edge1	FALSE
BOOL	test.edge2	FALSE
BOOL	test.in1	FALSE
BOOL	test.in2	FALSE

Setting the Project to Run Mode

To run the project:

1. On the **Runtime** menu, click **Run Project.**
2. Click **Continue** to run the project. This is the default option, even if you have not run the project before, because the programs were downloaded and then paused.
3. Note the values for count1 and count2 are both equal zero.
4. Double-click the value of in1 in the Watch window. The **Modify Value** dialog box appears. The default new value to write to the in1 is TRUE.

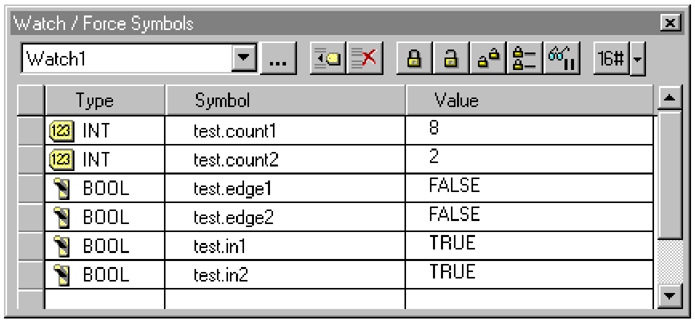


Modify Value

test.in1

New Value: [✓ TRUE] FALSE

5. Click **Write** to set in1 to TRUE. Note that count1 is incremented by one. Note also that count2 remains equal to zero since it is controlled by the other, independently executing function block instance.
6. Repeat steps 4-5 several times and note that the value of count1 increases.
7. Do the same for in2 and observe that count2 is incremented.



Watch / Force Symbols

Watch1

Type	Symbol	Value
INT	test.count1	8
INT	test.count2	2
BOOL	test.edge1	FALSE
BOOL	test.edge2	FALSE
BOOL	test.in1	TRUE
BOOL	test.in2	TRUE

Function Block Example: Monitoring the Variables

Additional Characteristics of Function Blocks

The following points characterize the function block.

- A Structured Text program is placed in the project hierarchy under programs. A function block template is created in the hierarchy, but you must create an instance in the symbol table.
- Although you can "Save As" programs to make copies of the code, you must edit the code to point to different variables. Function blocks have inputs and outputs and you can make many “copies” (instances) and pass different parameters into the same code.
- A Structured Text program executes every scan (assuming it is a normal priority program).
- A function block executes whenever it is called from a Structured Text program, an SFC program, or another function block.
- Because the function block instance is created in the symbol table, you can treat it like a user-defined data type with associated code.

  A simple symbol, such as a Boolean for example, has no code associated with it. You must change its values elsewhere in the project. A user-defined symbol allows you to define the variables in the structure, but it has no code associated with it either.

Developing a Function

This section describes how to create a new function. The following is the general procedure that you will follow:

Create a new function block type and enter its code.

Define the parameters for the function block type.

Add an instance of the function block to the project.

Create a new Structured Text program and enter code that calls the function block for execution.

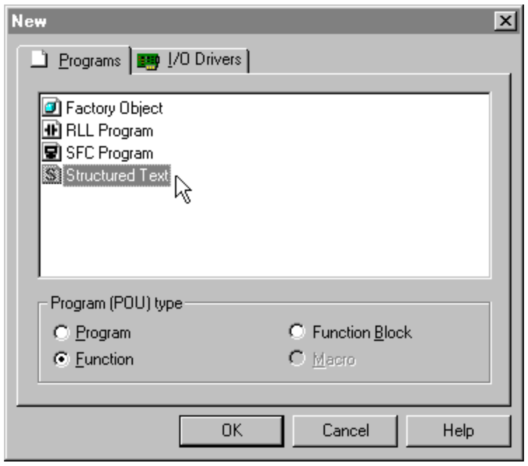
Note The figures in this section are based on a new project with no other programs. If you have already created an RLL program, for example, you may observe some minor differences in the dialog boxes.

Creating a New Function

To begin developing a function:

1. On the **File** menu, click **New**.

   The New dialog box appears.

   Function Example: New Dialog Box

2. Click **Structured Text** Program.
3. Click **Function** to select the Program Type. Then click OK.

   The **Save As** dialog box appears.

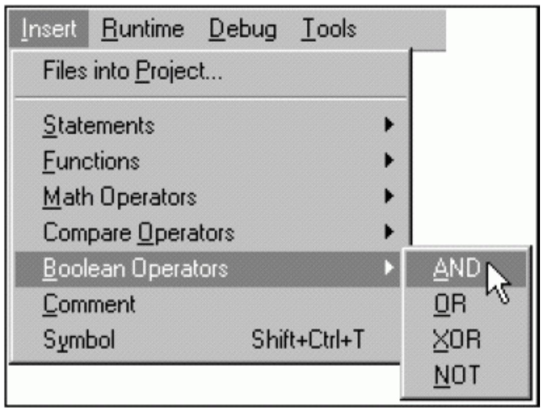
4. Enter a name for the function (up to 31 characters) and click Save.This example uses calc_add.stl for the name. The default extension .stl is appended when the file is saved in your project.

   The Structured Text editor displays a new Structured Text file ready for editing. To begin editing the function, see "Entering Function Code."

For a detailed description of the Structured Text editor and toolbar items, see the "Using the Structured Text Editor" chapter.

Entering Function Code

You can type Structured Text code directly into the editor window or by making selections from the **Insert** menu, which is shown in the following figure.



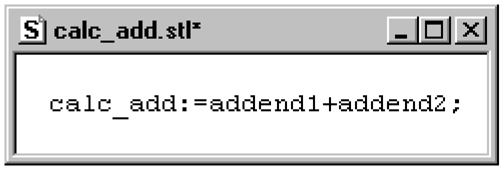
Function Example: Selecting STL Functions

To enter the example code:

Note You can copy the code from the online manual, paste it directly into the editor window, and avoid typing it in manually.

1. Type the following lines of code:
   `calc_add:=addend1+addend2;`

   The following figure shows the contents of the Structured Text editor after the code is entered:

   ```
   calc_add:=addend1+addend2;
   ```

   Function Example: Entering Code

   The asterisk by the program name in the window title bar indicates that the program has not been saved; or when the program is executing on the runtime engine, that this is not the same version because editing changes have been made.

2. On the **File** menu, click **Save.**

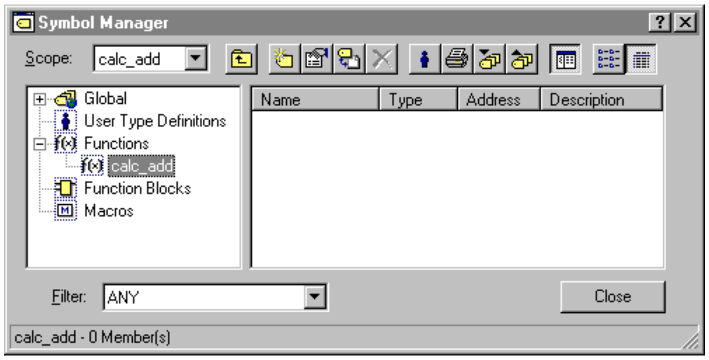
   To specify the data type of the return value for the function, see "Specifying Return Value Data Type."

Specifying Return Value Data Type

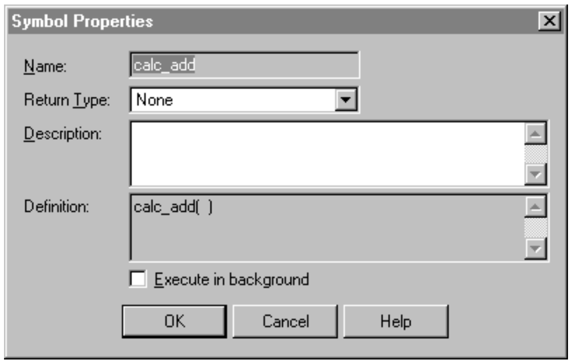
You specify the return type for a function in the Symbol Manager.

To specify the return data type for the function:

1. On the **Tools** menu, click **Symbol Manager.** The Symbol Manager appears.



2. Right-click the function and select **Properties.** The **Symbol Properties** dialog box appears.



Name: calc_add
Return Type: None
Description:
Definition: calc_add( )

Function Example: Return Value Properties

3. Select REAL as the data type in the **Return Type** field and enter an optional description for the function.

   Functions only return simple data types. They cannot return arrays, structures, or function blocks.

4. Leave the **Execute in Background** checkbox clear and click **OK** to save your work.

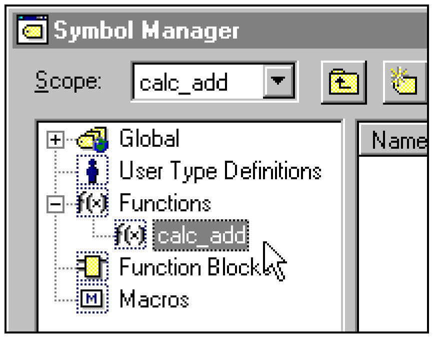
   To define the parameters used by the function, see "Creating Function Parameters."

Creating Function Parameters

You define the input and output parameters and variables for a function in the Symbol Manager. Function variables are local to the function and cannot be referenced elsewhere in the project, except within the context of the function call. Within the function, input parameters are read only. Output parameters must be assigned values through an assignment statement.

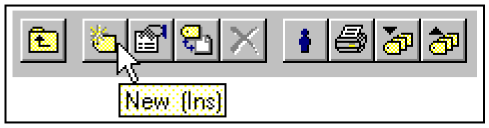
To define the parameters for the function:

1. On the **Tools** menu, click **Symbol Manager.**
2. Click the function to select it as shown in the following figure.

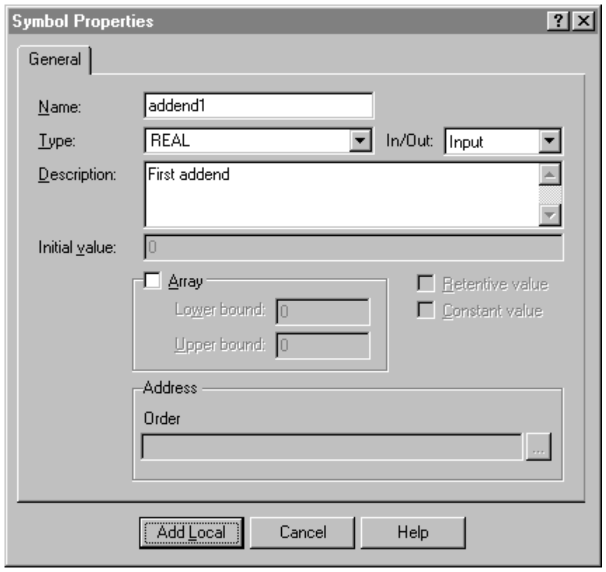


Function Example: Selecting the Function

3. Click **New** on the Symbol Manager toolbar.



The **Symbol Properties** dialog box appears.



Symbol Properties

General

Name: addend1

Type: REAL In/Out: Input

Description: First addend

Initial value: 0

Array

Lower bound: 0

Upper bound: 0

Retentive value

Constant value

Address

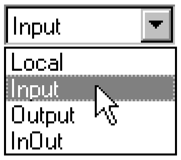
Order

Add Local Cancel Help

4. Enter addend1 for the name of the first parameter into the **Name** field.

5. Select REAL as the data type in the **Type** field.
6. Enter the optional description into the **Description** field. This example uses the following text:

   First addend.
7. Choose Input Parameter in the **In/Out** field, as shown below.

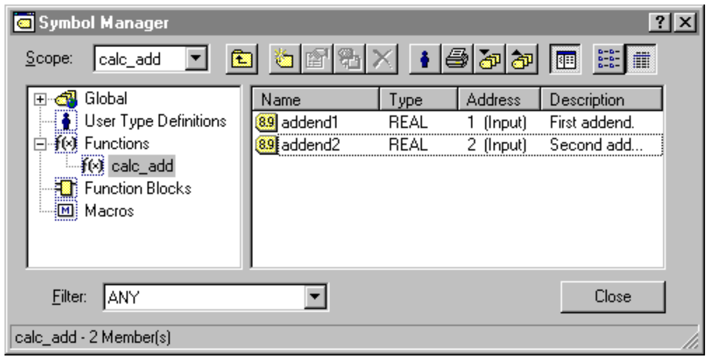


8. Click **Add Local** to complete the definition for this parameter.
9. Before closing the Symbol Manager, repeat steps 3-8 to add the other input parameter used by the example function block. Both are listed in the table below. Enter them in the order shown.
10. Click **Close** to close the Symbol Manager.

The parameters used by the function are listed in the following table.

Name	Data Type	In/Out	Description
addend1	REAL	Input	First addend.
Addend2	REAL	Input	Second addend.

The following figure shows the contents of the Symbol Manager after the two parameters have been defined. Although it does not matter for this example function, which only adds two numbers, the order of the parameters in the Symbol Manager is important. Parameter order is indicated in the **Address** field, as shown below.



Name	Type	Address	Description
addend1	REAL	1 (Input)	First addend.
addend2	REAL	2 (Input)	Second add...

Function Example: Parameters

If you enter parameters out of order, you can change their order in the Symbol Manager.

To change the order of parameters:

1. Right-click the parameter.
2. Click **Decrease Address** or **Increase Address** to change the order of the parameter in the list.

Creating the Calling Program

A function does not run automatically, but rather must be called by a program. This section describes how to create a Structured Text program to call the example function.

To create a Structured Text Program:

1. 1. Follow the procedure that you used to create a function, as described in "Creating a New Function."

   On the **File** menu, click **New.**

   Click **Structured Text Program.**

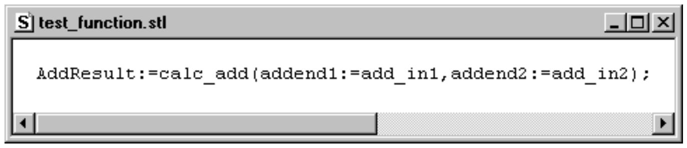
   Click **Program** to select the Program Type. Then click **OK.**

   Enter a name for the program (up to 31 characters) and click **Save**.This example uses test_function.stl for the name.

   The Structured Text editor displays a new Structured Text file ready for editing.

2. Enter the following code for the program.
   `AddResult:=calc_add(addend1:=add_in1,addend2:=add_in2);`

   The following figure shows the contents of the Structured Text editor after the code is entered:

   ```
   AddResult:=calc_add(addend1:=add_in1,addend2:=add_in2);
   ```

   Function Example: Code for Calling Program

3. On the **File** menu, click **Save.** Do not close the program.

   To define the variables used by the calling program, see "Creating Variables for the Calling Program."

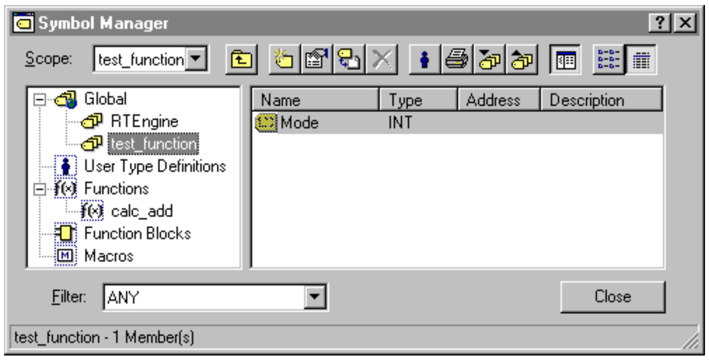
Creating Variables for the Calling Program

This section describes how to define variables for the program that calls the function.

1. If you have closed the calling program, double-click the program name to open it.

   Note that it is not necessary for the program to be open when you define its variables. However, when the program is open, any local variables that you define are associated with the program by default.

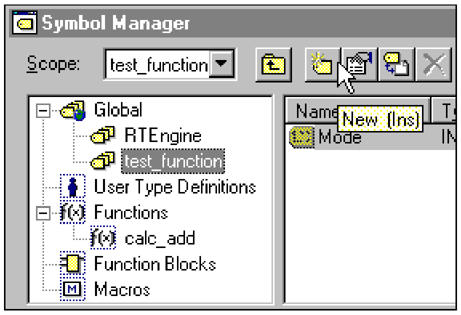
2. Click **Symbol Manager** on the **Tools** menu. The Symbol Manager appears.



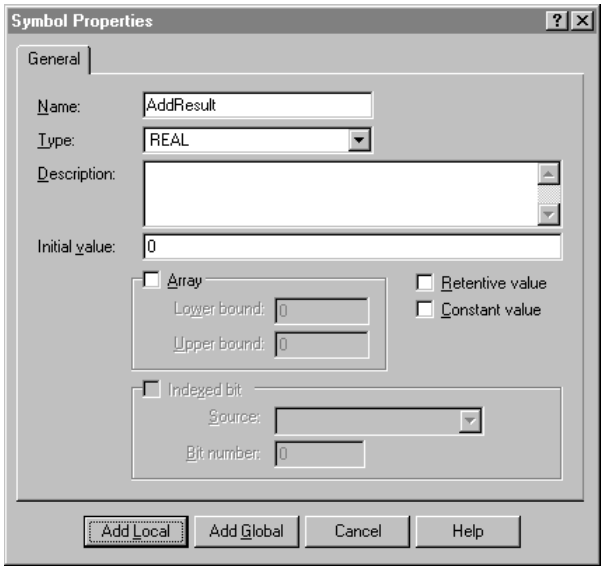
Name	Type	Address	Description
Mode	INT		

test_function - 1 Member(s)

3. Click **New** on the toolbar.



The **Symbol Properties** dialog box appears.



Name: AddResult
Type: REAL
Description:
Initial value: 0
Retentive value
Constant value

4. Enter `AddResult` in the **Name** field.
5. Click **REAL** in the **Type** field.
6. Click **Add Local** to add the new variable name to the Symbol Manager as a local variable.
7. Before closing the Symbol Manager, repeat steps 2-6 to add the remaining variables used by the calling program. All the variables are listed in the table below. Order does not matter for these variables.

   Continue the example by downloading and running the project. See "Calling and Running the Function."

The variables used by the calling program are listed in the following table.

Name	Data Type
AddResult	REAL
add_in1	REAL
add_in2	REAL

Calling and Running the Function

This section describes how to run the function. The following is the general procedure that you will follow:

Download the program.

Add the variables to the Watch Window.

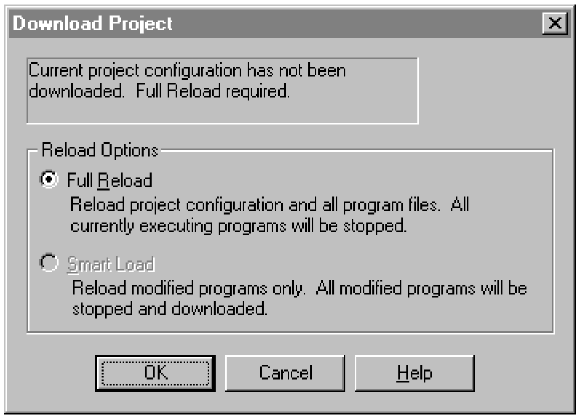
Set the program to Run mode.

Note that it is necessary to download and run the project because all the code, the calling program and the function, must be loaded to the runtime engine.

Downloading the Project

To download the project:

1. On the **Runtime** menu, click **Connect.** This connects the Development environment to the runtime engine. Ignore this step if you already connected to the runtime engine in order to run any of the other example programs.
2. On the **Runtime** menu, click **Download Project.** The **Download Project** dialog box appears.


Download Project

Current project configuration has not been downloaded. Full Reload required.

Reload Options

[✓ Full Reload]
Reload project configuration and all program files. All currently executing programs will be stopped.

Smart Load
Reload modified programs only. All modified programs will be stopped and downloaded.

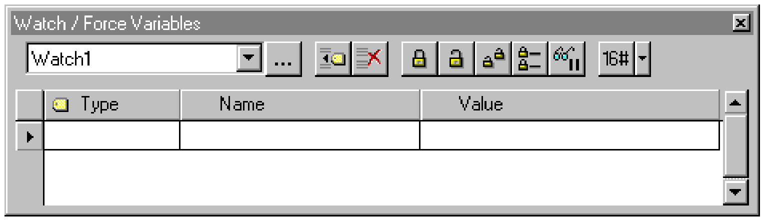
Function Example: Downloading the Project

3. Click **OK** to do a full reload of the runtime engine. The project is downloaded to the runtime engine and all programs are set to the Pause mode.

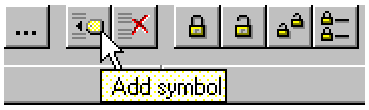
Adding Variables to the Watch Window

To add the program variables to the Watch Window:

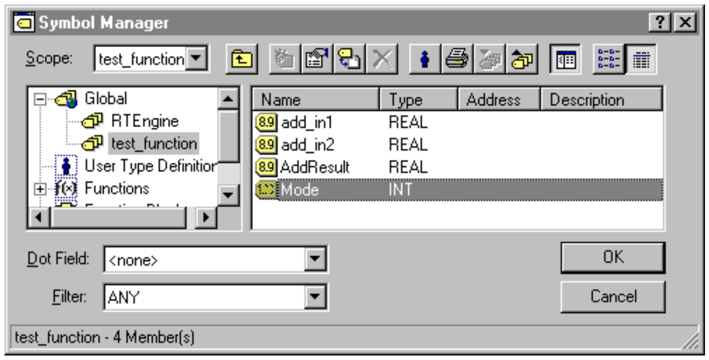
1. If the Watch Window is not visible, on the **View** menu, click **Watch/Force Variables.** The Watch window appears.



2. To add the variables to the Watch window, click **Add Symbol.**

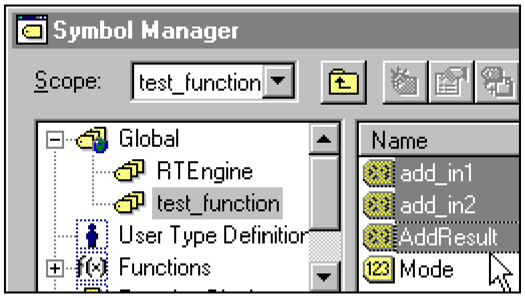


3. The **Symbol Manager** dialog box appears. The variables that you defined are displayed in the Symbol Manager.



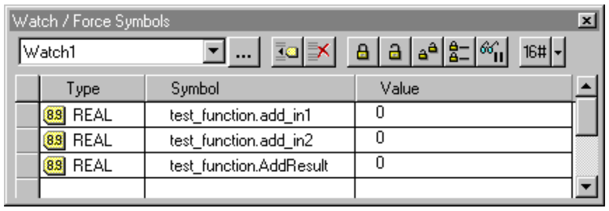
Name	Type	Address	Description
add_in1	REAL		
add_in2	REAL		
AddResult	REAL		
Mode	INT		

4. Click the following variable names to select them, and click **OK:** add_in1, add_in2, and AddResult. Use the **Ctrl** and **Shift** keys to select multiple items.



Function Example: Selecting Variables for Watch Window

The variables are added to the Watch window.

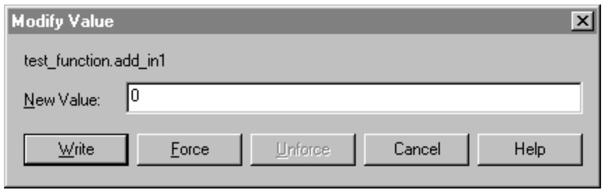


Type	Symbol	Value
REAL	test_function.add_in1	0
REAL	test_function.add_in2	0
REAL	test_function.AddResult	0

Setting the Project to Run Mode

To run the project:

1. On the **Runtime** menu, click **Run Project.**
2. Click **Continue** to run the project. This is the default option, even if you have not run the project before, because the programs were downloaded and then paused.
3. Note the values for add_in1 and add_in2 are both equal zero.
4. Double-click the value of add_in1 in the Watch window. The **Modify Value** dialog box appears. The default new value to write to the add_in1 is zero.

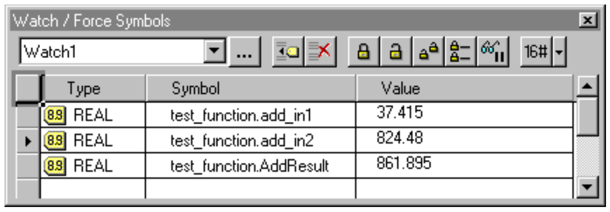


test_function.add_in1

New Value: 0

5. Enter 37.415 and click **Write** to enter the value in add_in1 to be added.
6. Enter 824.48 for the value for add_in2. The Structured Text program test_function, which is continually calling calc_add, stores the result of the addition to AddResult.

7. Note the result of the addition in AddResult.



Type	Symbol	Value
REAL	test_function.add_in1	37.415
REAL	test_function.add_in2	824.48
REAL	test_function.AddResult	861.895

Function Example: Monitoring the Variables

Additional Characteristics of Functions

The following points characterize the function.

- Functions allow you to call the same code from many locations in your project.
- Functions allow you to return a variable.
- Values for local variables defined in the function are not saved between function calls. Use function blocks if you want to save local variable values between calls.

Index

Wonderware®

InControl™ Function and Function Block Reference User's Guide

Revision H

Last Revision: January 2004

Invensys Systems, Inc.

Invensys Systems, Inc.
26561 Rancho Parkway South
Lake Forest, CA 92630 U.S.A.
(949) 727-3200
http://www.wonderware.com

Contents

C H A P T E R 1

Functions and Function Blocks

This chapter introduces the programming elements that you use in an RLL program.

Contents

Extensions to IEC 61131-3

This section describes the enhancements and other extensions to the IEC 61131-3 specification. InControl complies with IEC 61131-3 except where noted here.

Counter Parameters

The Preset Value and the Current Value parameters used in the RLL counter function blocks are DINT data types.

Math Functions

The following RLL math functions have inputs and outputs that accept any of the Any_Bit data types, except for the BOOL:

- ADD
- DIV
- SUB
- MUL
- MOD

Parameters

The parameters for Structured Text functions can be listed in any order as long as the formal parameter names are given as specified by IEC-61131-3.

Unsupported Functions

InControl does not support the following functions, which are defined in the IEC 61131-3 specification: LIMIT, MUX, SEL

Unsupported Function Blocks

InControl does not support the following function blocks, which are defined in the IEC 61131-3 specification: SR, RS, SEMA, EDGE_CHECK, RTC.

Additional Built-In Functions and Function Blocks

InControl provides the following functions and function blocks, which are not defined in the IEC 61131-3 specification: ARRAY_TO_STRING, STRING_TO_ARRAY, CLOSEFILE, COPYFILE, DELETEFILE, NEWFILE, OPENFILE, READFILE, REWINDFILE, WRITEFILE, MSGWND, ABORT_ALL.

Function/Function Block Groups

The InControl functions and function blocks are predefined algorithms that carry out single operations.

InControl Functions and Function Blocks

Group	Type	Description	Manual Page
Bitwise	AND	Computes the bitwise AND of two numbers.	19
	NOT	Computes the bitwise complement of a number.	87
	OR	Computes the bitwise OR of two numbers.	90
	ROL	Rotates the input left by a specified number of bits.	105
	ROR	Rotates the input right by a specified number of bits.	106
	SHL	Shifts the input left by a specified number of bits.	107
	SHR	Shifts the input right by a specified number of bits.	108
	XOR	Computes the bitwise Exclusive OR of two numbers.	136
Comparison	EQ	Tests two inputs for equality.	54
	GE	Tests if first input is greater than or equal second input.	60
	GT	Tests if first input is greater than second input.	61
	LE	Tests if first input is less than or equal second input.	67
	LT	Tests if first input is less than second input.	72
	NE	Tests two inputs for inequality.	8

Group	Type	Description	Manual Page
Conversion	ARRAY_TO_STRING	Takes a byte array input and stores the bytes as characters in a string.	20
	BCD_TO_INT	Converts a Binary -Coded Decimal (BCD) input to an ANY_INT value.	25
	DATE_TO_REAL	Converts a DATE data type input to an ANY_REAL value.	45
	DATE_TO_STRING	Converts a DATE data type input to a string.	47
	INT_TO_BCD	Converts an integer to the equivalent Binary-Coded Decimal (BCD) representation of the value.	64
	INT_TO_REAL	Converts an ANY_INT input to an ANY_REAL value.	65
	INT_TO_STRING	Converts an ANY_INT input to a string.	66
Conversion	REAL_TO_DATE	Converts an ANY_REAL input to a DATE value.	9
	REAL_TO_INT	Converts an ANY_REAL input to an ANY_INT value.	9
	REAL_TO_STRING	Converts an ANY_REAL input to a string.	9
	REAL_TO_TIME	Converts an ANY_REAL input to a TIME value.	9
	STRING_TO_ARRAY	Takes a string input and stores the characters of the string in a byte array.	11
	STRING_TO_DATE	Converts an input string to a DATE value.	11
	STRING_TO_INT	Converts an input string to an ANY_INT value.	11
	STRING_TO_REAL	Converts a string input to an ANY_REAL value.	11
	STRING_TO_TIME	Converts a string input to a TIME value.	11
	TIME_TO_REAL	Converts a TIME input to an ANY_REAL value.	12
	TIME_TO_STRING	Converts a TIME input to a string.	12

Group	Type	Description	Manual Page
Counter	CTD	Counts events by decrementing by one.	35
	CTU	Counts events by incrementing by one.	38
	CTUD	Counts events up or down.	41
File	CLOSEFILE	Closes a file.	26
	COPYFILE	Copies a file.	32
	DELETEFILE	Deletes a file.	50
	NEWFILE	Creates a new file.	8
	OPENFILE	Opens an existing file.	8
	READFILE	Reads data from a file.	9
	REWINDFILE	Rewinds a file to the beginning.	10
	WRITEFILE	Writes data to a file.	13
Math	ABS	Computes the absolute value of a value.	1
	ADD	Adds two values.	1
	DIV	Divides one value by another.	5
	EXPT	Raises a value to the power specified by a second value.	5
	MAX	Determines the larger of two values.	73
	MIN	Determines the smaller of two values.	75
	MOD	Divides one value by another and stores the remainder.	76
	MOVE	Copies data from one location to another.	78
	MUL	Multiplies two values.	82
	NEG	Negates (inverts) the inputs.	84
	SQRT	Computes the square root of a value.	110
	SUB	Subtracts one value from another.	117
	TRUNC	Removes one or more of the least significant digits of an ANY_REAL data type.	132

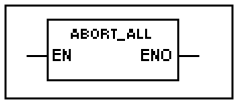
Group	Type	Description	Manual Page
String	CONCAT	Concatenates a string input to the end of another string.	3
	DELETE	Deletes characters from the middle of a string input.	4
	FIND	Searches for one string input within another.	5
	INSERT	Inserts a string input into another string.	6
	LEFT	Copies the leftmost characters from a string input.	6
	LEN	Stores the length of a string input.	6
	MID	Copies characters from the middle of a string input.	74
	MSGWND	Displays a message in the Output Window.	80
	READFILE	Replaces characters in a string input with another string input.	92
	RIGHT	Copies the rightmost characters from a string input.	104
Timer	TOF	Provides off-delay timing of events.	122
	TON	Provides on-delay timing of events.	126
	TP	Activated by a pulse, provides off-delay timing of events.	129
Trig/Log	ACOS	Computes the arc cosine of a value.	16
	ASIN	Computes the arc sine of a value.	23
	ATAN	Computes the arc tangent of a value.	24
	COS	Computes the cosine of a value.	34
	EXP	Computes the natural log exponentiation of a value.	55
	LN	Computes the natural log of a value.	70
	LOG	Computes the log (base 10) of a value.	71
	SIN	Computes the sine of a value.	10
	TAN	Computes the tangent of a value.	11

Group	Type	Description	Manual Page
Trigger	ABORT_ALL	Aborts all programs that are running.	14
	F_TRIG	Turns on an output when triggered by a falling edge trigger.	58
	R_TRIG	Turns on an output when triggered by a rising edge trigger.	9

ABORT_ALL

The ABORT_ALL function block aborts all programs running in the runtime engine.

Graphical representation:



Structured text syntax:

```
ABORT_ALL();
```

Operation is as follows:

- ABORT_ALL aborts all programs (RLL, SFC, Structured Text, etc.), changes the runtime engine monitor icon to the error condition, and sets the runtime engine to the Fault mode.
- No other logic following the ABORT_ALL is executed.
- The EN BOOL parameter is used only in the graphical languages to enable the function block to execute. ENO follows EN unless an error condition occurs within the function block.

WARNING! The ABORT_ALL function block stops all programs in a project. An unplanned stop of all programs can cause unpredicted operation by output devices which can result in injury or death to personnel and/or damage to equipment. Design your program code very carefully so that the ABORT_ALL is only executed under carefully controlled conditions.

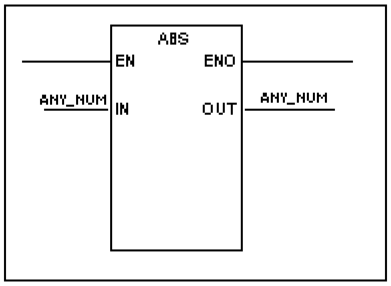
Other Trigger Function Blocks

F_TRIG	R_TRIG

ABS

The ABS function calculates the absolute value of the input.

Graphical representation:



Structured text syntax:

```
<OUT> := ABS(<IN>);
```

Parameter	Type	Description
IN	ANY_NUM	Contains the value for which the absolute value is calculated. Any SINT, INT, DINT, REAL, or LREAL constant or variable.
OUT	ANY_NUM	Contains the absolute value of the input. Any SINT, INT, DINT, REAL, or LREAL variable.

Operation is as follows:

- ABS takes the absolute value of the input and stores the result in the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

Examples of the ABS operation:

ABS(14) returns 14.

ABS(-7.5) returns 7.5.

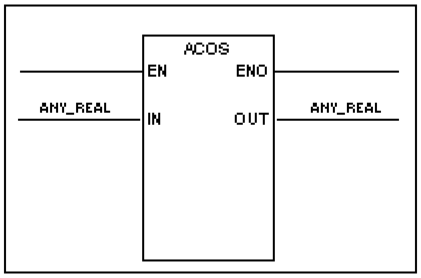
Other Math Functions

ADD	DIV	EXPT
MAX	MIN	MOD
MOVE	MUL	NEG
SQRT	SUB	TRUNC

ACOS

The ACOS function calculates the arc cosine of the input. The result is in radians.

Graphical representation:



Structured text syntax:

```
<OUT> := ACOS(<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the value for which the arc cosine is calculated. Any REAL or LREAL constant or variable (-1.0 to + 1.0).
OUT	ANY_REAL	Contains the arc cosine of IN in radians. Any REAL or LREAL variable.

Operation is as follows:

- If the input is within range (-1 to +1), ACOS stores the arc cosine of the input to the output variable in radians.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Examples of the ACOS operation:

ACOS(0) returns pi/2 (1.570...).

ACOS(1) returns 0.

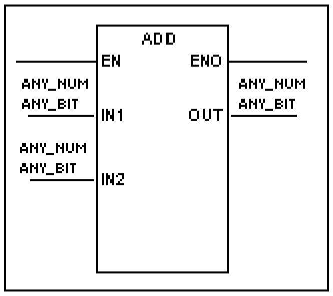
Other Trig/Log Functions

ASIN	ATAN	COS
EXP	LN	LOG
SIN	TAN	

ADD

The ADD function sums the values of two inputs.

Graphical representation:



Structured text syntax:

```
<OUT> := <IN1> + <IN2>;
```

Parameter	Type	Description
IN1	ANY_NUM ANY_BIT [1]	Contains the first value to be added. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
IN2	ANY_NUM ANY_BIT [1]	Contains the second value to be added. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
OUT	ANY_NUM ANY_BIT [1]	Contains the sum of the addition of IN1 and IN2. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD variable.

1 BOOL data types are not allowed.

Operation is as follows:

- ADD adds the two inputs and stores the sum to the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

You can also use this function with arrays. For example, you can add a number to every element in an array with the following line:

Array1 := Array2 +1;

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

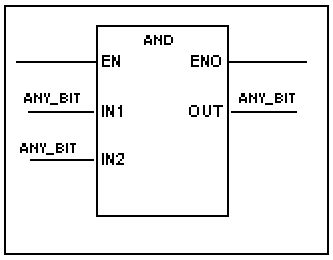
Other Math Functions

ABS	DIV	EXPT
MAX	MIN	MOD
MOVE	MUL	NEG
SQRT	SUB	TRUNC

AND

The AND function does a bitwise logical AND of two values.

Graphical representation:



Structured text syntax:

```
<OUT> := <IN1> AND <IN2>;
```

Parameter	Type	Description
IN1	ANY_BIT	Contains the first value to be ANDed. Any BOOL, BYTE, WORD, DWORD constant or variable.
IN2	ANY_BIT	Contains the second value to be ANDed. Any BOOL, BYTE, WORD, DWORD constant or variable.
OUT	ANY_BIT	Contains the result of ANDing IN1 and IN2. Any BOOL, BYTE, WORD, DWORD variable.

Operation is as follows:

- AND does a logical AND on each bit of the two inputs.
- AND stores the result of the AND operation to the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

You can also use this function with arrays as shown in the examples below.

Examples of the AND operation:

2#0011 AND 2#0101 returns 2#0001.

TRUE AND FALSE returns FALSE.

Array1 := Array2 AND #16FFFE;

Array1 := Array1 AND Array2;

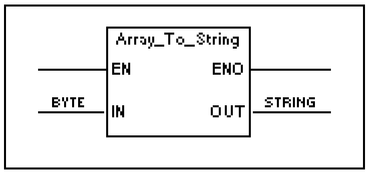
Other Bitwise Functions

NOT	OR	ROL
ROR	SHL	SHR
XOR		

ARRAY_TO_STRING

The ARRAY_TO_STRING function converts an input array of bytes to a fixed-length string.

Graphical representation:



Structured text syntax:

```
ARRAY_TO_STRING(OUT := <OUT> ,IN := <IN>);
```

Parameter	Type	Description
IN	BYTE	Specifies the array of bytes to be converted. Any array of bytes containing valid STRING character codes. Values are decimal codes.
OUT	STRING	Contains the result of the conversion of the array of bytes to a fixed-length string. Any STRING variable.

Operation is as follows:

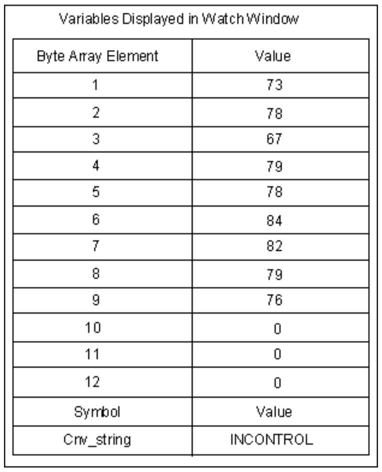
- ARRAY_TO_STRING converts each byte in the input array to an ASCII character, stores each character to the output variable.
- The length of the string in OUT is the same as the size of the array.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

The following is an example of the ARRAY_TO_STRING operation:

The input variable = byte_array

The output variable = Cnv_string

If byte_array consists of 12 elements with the values shown in the following figure, then Cnv_string contains 'InControl'.



Variables Displayed in Watch Window

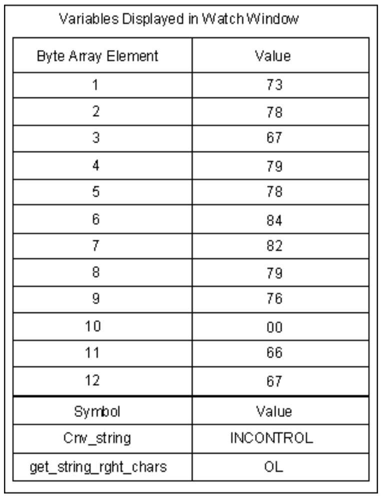
Byte Array Element	Value
1	73
2	78
3	67
4	79
5	78
6	84
7	82
8	79
9	76
10	0
11	0
12	0
Symbol	Value
Cnv_string	INCONTROL

ARRAY_TO_STRING Example 1

Because the string is of fixed length, the output of the ARRAY_TO_STRING function is the same length as the size of the array. If the RIGHT function is executed on the output of the ARRAY_TO_STRING, the count begins at the rightmost element in the output. In the following example, the RIGHT function is executed after the ARRAY_TO_STRING,

IN = Cnv_string
L = 2
OUT = get_string_rght_chars.

The string that is created contains OL when elements 8-9 of the array contain 82 and 79, respectively. See the following figure.



Variables Displayed in Watch Window

Byte Array Element	Value
1	73
2	78
3	67
4	79
5	78
6	84
7	82
8	79
9	76
10	00
11	66
12	67
Symbol	Value
Cnv_string	INCONTROL
get_string_rght_chars	OL

ARRAY_TO_STRING Example 2

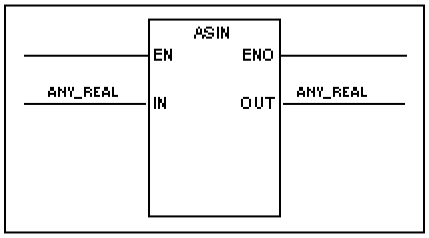
Other Conversion Functions

BCD_TO_INT	DATE_TO_REAL	DATE_TO_STRING
INT_TO_BCD	INT_TO_REAL	INT_TO_STRING
REAL_TO_DATE	REAL_TO_INT	REAL_TO_STRING
REAL_TO_TIME	STRING_TO_ARRAY	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

ASIN

The ASIN function calculates the arc sine of the input. The result is in radians.

Graphical representation:



Structured text syntax

```
<OUT> := ASIN(<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the value for which the arc sine is calculated. Any REAL or LREAL constant or variable (-1.0 to + 1.0).
OUT	ANY_REAL	Contains the arc sine of IN in radians. Any REAL or LREAL variable.

Operation is as follows:

- If the input is within range (-1 to +1), ASIN stores the arc sine of the input to the output variable in radians.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Examples of the ASIN operation:

ASIN(0) returns 0.

ASIN(1)returns pi/2 (1.570...).

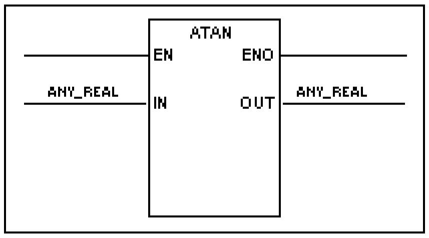
Other Trig/Log Functions

ACOS	ATAN	COS
EXP	LN	LOG
SIN	TAN	

ATAN

The ATAN function calculates the arc tangent of the input. The result is in radians.

Graphical representation:



Structured text syntax:

```
<OUT> := ATAN(<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the value for which the arc tangent is calculated. Any REAL or LREAL constant or variable.
OUT	ANY_REAL	Contains the arc tangent of IN in radians. Any REAL or LREAL variable.

Operation is as follows:

- If the input is within range of the selected data type, ATAN stores the arc tangent of the input to the output variable in radians.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Examples of the ATAN operation:

ATAN(0) returns 0.

ATAN(1) returns pi/4.

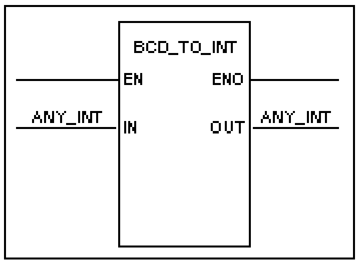
Other Trig/Log Function Blocks

ACOS	ASIN	COS
EXP	LN	LOG
SIN	TAN	

BCD_TO_INT

The BCD_TO_INT function converts a Binary-Coded Decimal (BCD) input to an ANY_INT value.

Graphical representation:



Structured text syntax:

```
<OUT> := BCD_TO_INT (<IN>);
```

Parameter	Type	Description
IN	ANY_INT	Contains the value to convert. A BCD number, variable or expression that resolves to an ANY_INT data type (SINT, INT, DINT, BYTE, WORD, DWORD).
OUT	ANY_INT	Contains the result of the conversion. An ANY_INT variable data type (SINT, INT, DINT, BYTE, WORD,DWORD).

Operation is as follows:

- BCD_TO_INT converts the integer representation of the BCD input to an integer with the value of the BCD input.
- If the input is an invalid BCD number, the output variable is set to -1.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the BCD_TO_INT function.

BCD_TO_INT(16#321) returns 10#321

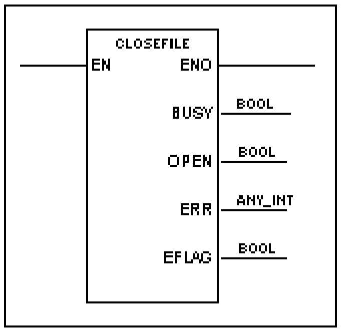
Other Conversion Functions

ARRAY_TO_STRING	DATE_TO_REAL	DATE_TO_STRING
INT_TO_BCD	INT_TO_REAL	INT_TO_STRING
REAL_TO_DATE	REAL_TO_INT	REAL_TO_STRING
REAL_TO_TIME	STRING_TO_ARRAY	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

CLOSEFILE

The CLOSEFILE function closes a file that has been opened by the OPENFILE or the NEWFILE functions. The CLOSEFILE is one of eight functions that do file operations. Note that these functions are not designed for high-speed I/O execution or data transfers of large blocks of information.

Graphical representation:



Structured text syntax:

```
CLOSEFILE (<file control block name>);
```

Parameter	Type	Description
File Control Block Name	FILE	Name of the file control block that handles operations for this file.
BUSY [1]	BOOL	Indicates the file control block is busy. Any BOOL variable.
OPEN [1]	BOOL	Indicates the file has been opened. Any BOOL variable.
ERR [1]	ANY_INT	Contains the error code if a file operation error occurs. Any SINT, INT, DINT, BYTE, WORD, DWORD variable.
EFLAG [1]	BOOL	Indicates a file operation error has occurred. Any BOOL variable.
1 Entries in the output fields are optional. However, for each field there is a default file control variable. As you design your program, you must use these output variables to handle file control. These fields reflect outputs in the specified file control block and are not actual parameters to the CLOSEFILE function.		

Operation is as follows:

- CLOSEFILE closes the file that is associated with the control block, which you specify in the File Control Block Name field.

  Note that you do not designate the file by its file name, only by the file control block. All file functions that operate on the same file must use the same File Control Block name.

- If an error occurs, the file error variable is set to TRUE and a message appears in the Output window and the Wonderware Logger. The graphical output ENO is set to FALSE.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

All the file-type functions use file control block variables to handle file control. Three of these variables specify status of a file after it is open: read/write, whether data can be appended, and whether other applications can access the file. Eight variables provide a means of monitoring errors, whether a file is in use, when an operation is completed, etc. You can use any of these variables in an expression, contact or coil instead of a symbol of the same type. To reference a variable, enter the control block name followed by a period and the specific variable name. For example, **FILEA.BUSY** refers to the file control block variable **BUSY** for the file referenced by control block **FILEA**.

For the three variables that specify file status after it is open, you can specify initial values for the variables in the **Symbol Properties** dialog box in the Symbol Manager. If you prefer, you can use MOVE functions to assign values to the variables before opening the file. These three input variables are listed in the following table.

File Control Input Variables

Variable	Description	
fcb.ACCESS [1]	Byte variable specifies read/write status of the file after it opens.	
	FileAccess.ReadWrite	= (default) file is open for read/write operations.
	FileAccess.Read	= file is open for read-only operations.
	FileAccess.Write	= file is open for write-only operations.
fcb.APPEND [1]	Boolean variable specifies whether data can be appended to the file after it opens. Only valid when file is open with write status. That is, the ACCESS variable = 0 or 2.	
	TRUE	= data will be appended to the file.
	FALSE	= (default) data cannot be appended to the file.
fcb.SHARE [1]	Byte variable specifies how file can be accessed after it is open.	
	FileShare.ReadWrite	= (default) other applications can access the file for read-write operations.
	FileShare.Read	= other applications can access the file for read-only operations.
	FileShare.Write	= other applications can access the file for write-only operations.
	FileShare.None	= other applications cannot access the file.

1 These variables are read and take effect only when the FILEOPEN and FILENEW function blocks are executed.

The seven output variables that handle file operations are listed in the following table.

File Control Output Variables

Variable	Description
fcb.OPEN [1]	Boolean variable indicates the file has been opened. The system sets the File Open variable to TRUE when the file is open.
fcb.BUSY [1]	Boolean variable indicates that the file is being accessed. The system sets the File Control Busy variable to TRUE when the file is being accessed by another file function. If you attempt to execute a file type function while this variable is TRUE, an error occurs (error code 15).
fcb.EFLAG [1]	Boolean variable indicates when an error occurs. If an error occurs during a file operation, the system sets the File Error variable to TRUE. This variable is not reset automatically; the program must reset the variable. You can also reset it manually through the Watch window. A file type function cannot execute while this variable is TRUE. The program does not go into Fault mode when an error occurs.
fcb.RDN	Boolean variable indicates that a read operation has been completed. The system sets the File Read Done variable to TRUE when the read operation is finished.
fcb.WDN	Boolean variable indicates that a write operation has been completed. The system sets the File Write Done variable to TRUE when the write operation is finished.
fcb.EOF	Boolean variable indicates that the system encountered an End Of File. The system sets the End Of File variable to TRUE when it encounters the EOF.
fcb.ERR [1]	Integer variable contains the error code if an error occurs. If an error occurs during a file operation, the system writes an error code to the File Error Code integer. The table that follows lists the error codes.
1 Entries in these output fields for graphical languages are optional. However, for each field there is a default file control variable. As you design your program, you must use these output variables to handle file control. Use either your own meaningful names, or the default variable names.	

The following table lists the error codes for the file type function blocks.

File Control Error Codes

Error Code	Error Description
15	File control block is busy.
16	No file name specified.
17	File has not been opened.
18	File not found.
19	Disk full.
20	Read failed.
21	File copy failed.
22	Write failed.
25	File close failed.
26	File already exists.
27	File is open.
28	File is read only.
29	File open failed.
30	General error.
31	Reached end of file.
32	Share violation. Another operation has locked the file.
33	Access denied. Can occur if you request one type of access and use another.
34	Not enough memory. The maximum size of a field of data that you can read/write is 1 KB.
35	Bad data. Can occur if specified data types are mismatched.

You can do only one file operation for each File Control Block at a time. Note that file control I/O operations take place asynchronously to program execution.

The following is an example of the CLOSEFILE function.

```
CLOSEFILE (FCB:= datrpt);
```

The system closes the file associated with the file control block called datrpt.

For an example that uses the CLOSEFILE with several other File procedures, see "WRITEFILE."

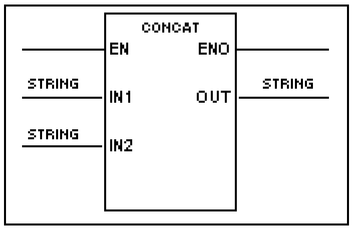
Other File Functions.

COPYFILE	DELETEFILE	NEWFILE
OPENFILE	READFILE	REWINDFILE
WRITEFILE		

CONCAT

The CONCAT function concatenates an input string to the end of another input string.

Graphical representation:



Structured text syntax:

```
<OUT> := CONCAT (<IN1>, <IN2>);
```

Parameter	Type	Description
IN1	STRING	Specifies the first string. Any valid STRING character or STRING variable.
IN2	STRING	Specifies the second string. Any valid STRING literal or STRING variable.
OUT	STRING	Contains the result of the concatenation of the strings. Any STRING variable.

Operation is as follows:

- CONCAT concatenates the string in the second input to the end of the string in the first input.

  CONCAT stores the result to the output variable
- If the sum of the lengths of the two strings is greater than 2048, the output variable is set to an empty string.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the CONCAT operation:

CONCAT('AB', 'SMITH') returns 'ABSMITH'.

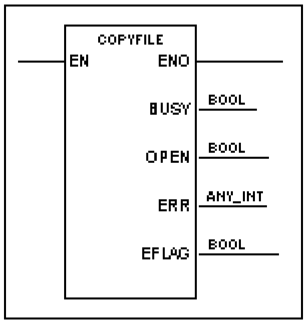
Other String Functions

DELETE	FIND	INSERT
LEFT	LEN	MID
MSGWND	REPLACE	RIGHT

COPYFILE

The COPYFILE function copies an existing file. The COPYFILE is one of eight function that do file operations. Note that these functions are not designed for high-speed I/O execution or data transfers of large blocks of information.

Graphical representation:



Structured text syntax:

```
COPYFILE (FCB:= <file control block name>,OUT:=
   <destination file name>, IN:= <source file name>);
```

Parameter	Type	Description
FCB	FILE	Name of the file control block that handles operations for this file.
IN	STRING	Name of the file to be copied. The default path is the same as for the runtime engine. Any STRING variable or file name.
OUT	STRING	Name of the file to which the source is copied. Any STRING variable or file name.
BUSY [1]	BOOL	Indicates the file control block is busy. Any BOOL variable.
OPEN [1]	BOOL	Indicates the file has been opened. Any BOOL variable.
EFLAG [1]	BOOL	Indicates a file operation error has occurred. Any BOOL variable.
ERR [1]	ANY_INT	Contains the error code if a file operation error occurs. Any SINT, INT, DINT, BYTE, WORD, DWORD variable.
1 Entries in the output fields are optional. However, for each field there is a default file control variable. As you design your program, you must use these output variables to handle file control. These fields reflect outputs in the specified file control block and are not actual parameters to the COPYFILE function. For a detailed description of the file control variables, see "CLOSEFILE."		

Operation is as follows.

- COPYFILE copies the source file to a new file and assigns it the destination file name. The default file path is the same as that of the runtime engine.
- The file control variables handle access to the file and error conditions, as described for the "CLOSEFILE" function block. All file function blocks that operate on the same file must use the same File Control Block name.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.
- If an error occurs, the file error variable is set to TRUE and a message appears in the Output window and the Wonderware Logger. The graphical output ENO is set to FALSE. Attempting to copy over an existing file generates an error (error code 26). For a complete list of the error codes, see "CLOSEFILE."

You can do only one file operation for each File Control Block at a time. Note that file control I/O operations take place asynchronously to program execution.

The following is an example of the COPYFILE function.

```
COPYFILE (FCB:= datrpt, OUT:= "newdatcopy" IN:=
   "olddatcopy");
```

The system makes a copy of the file called olddatcopy and names it newdatcopy.

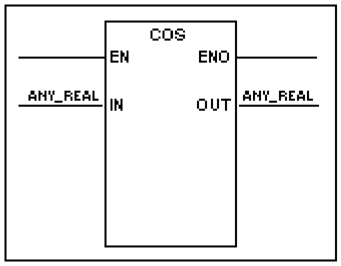
Other File Functions

CLOSEFILE	DELETEFILE	NEWFILE
OPENFILE	READFILE	REWINDFILE
WRITEFILE		

COS

The COS function calculates the cosine of the input, which must be in radians.

Graphical representation:



Structured text syntax:

```
<OUT> := COS(<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the value in radians for which the cosine is calculated. Any REAL or LREAL constant or variable.
OUT	ANY_REAL	Contains the cosine of IN. Any REAL or LREAL variable.

Operation is as follows:

- If the input (radians) is within range of the selected data type, COS stores the cosine of the input to the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Examples of the COS operation:

COS(0) returns 1.

COS(pi/2) returns 0.

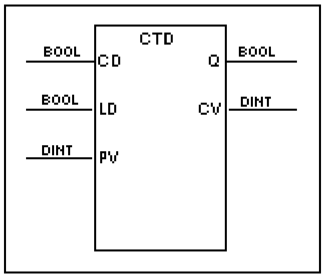
Other Trig/Log Functions

ACOS	ASIN	ATAN
EXP	LN	LOG
SIN	TAN	

CTD

The CTD function block counts recurring events down from a preset count, turning on the output when the current count is equal to or less than zero.

Graphical representation:



Structured text syntax:

```
<CTD Name>(CD:=<count down>, LD:=<load>, PV:=<preset
   value>, EN:=<enable>,Q:=<output>, CV:=<current value>,
   ENO:=<enable output>);
```

Parameter	Type	Description
CTD Name	CTD	Unique name for the counter.
CD	BOOL	Input decrements the counter when CD transitions from FALSE to TRUE. Any BOOL variable.
LD	BOOL	Loads the counter with the Preset Value. Any BOOL variable.
PV	DINT	Contains the value at which the CTD begins to count. Any DINT constant or variable.
EN	BOOL	Enables the counter. Any BOOL variable.
Q	BOOL	Output changes to TRUE when CV = zero. Any BOOL variable.
CV	DINT	Contains the current count of the counter. Any DINT variable.
ENO	BOOL	Echoes EN. Any BOOL variable.

WARNING! Assigning the same function block name to different counters may cause unpredictable operation by the controller, which can result in death or injury to personnel and/or damage to equipment. Always use a unique name for each counter.

Operation is as follows:

- When the enable input EN is TRUE the counter is enabled.

  When EN is FALSE, the counter is not enabled and cannot begin counting.
- When load LD is set to TRUE, current value CV is set equal to preset value PV, and output Q is set to FALSE.

  The counter does not begin counting while LD is TRUE.
- The counter decrements CV by one each time the count down input CD transitions from FALSE to TRUE.
- When CV = zero, the counter rung output Q is set to TRUE.
- If EN is set to FALSE while the counter is counting, the counter freezes Q and CV in their current states until EN is set to TRUE again.
- Enable output ENO echoes the value of EN.

When the program is running, you can double-click a CTD to display the CTD dialog box. The box displays the current value of all function block inputs and outputs. You can also open the Watch window and enter counter variables that you want to observe at runtime.

You can use any of the CTD inputs and outputs in any expression, contact or coil instead of a symbol of the same type. To reference a CTD input or output, enter the function block name followed by a period and the specific input or output suffix. For example, CTD1.CD refers to the count down input of CTD1. All counter variables are listed in the following table.

Variable	**Reference Name**
CD	xxx.CD
LD	xxx.LD
PV	xxx.PV
EN	xxx.EN
Q	xxx.Q
CV	xxx.CV
ENO	xxx.ENO
Note xxx denotes the counter function block name.	

An example of the CTD timing diagram is shown in the figure at the end of this section. The sequence of events is listed below:

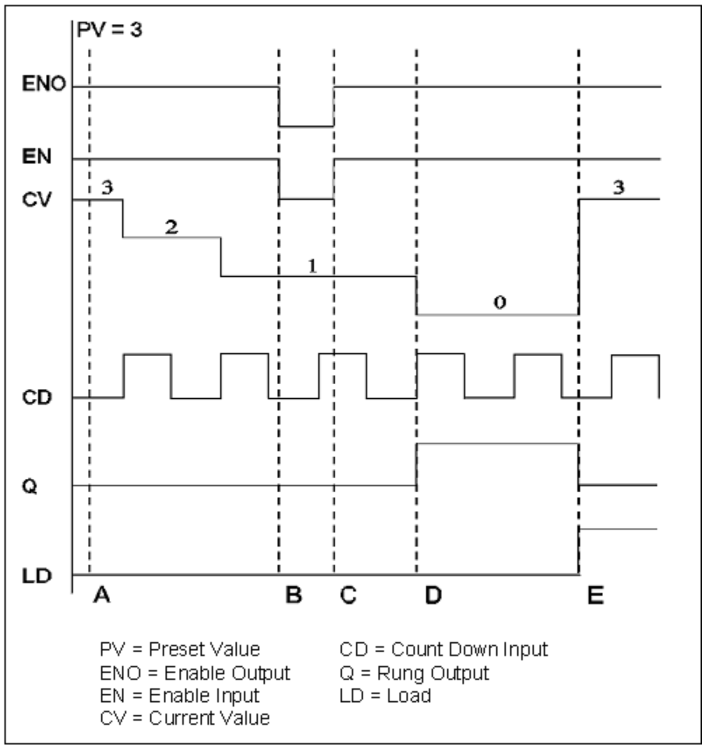
A. PV has been previously loaded with the preset value of 3.
EN and ENO have been previously been set to TRUE.
CV contains the current value of 3.

B. EN transitions from TRUE to FALSE, disabling the counter.
CV, which had been counting down as CD changed state, holds at 1.

C. EN transitions from FALSE to TRUE, re-enabling the counter.
CV resumes counting down at the next FALSE-to-TRUE transition of CD, and reaches 0.

D. Q transitions to TRUE when CV=0.

E. LD transitions from FALSE to TRUE, which loads the preset value of 3 into CV.
CV holds at 3 while LD is TRUE.
Q transitions to FALSE when CV contains the preset value.



CTD Timing Diagram

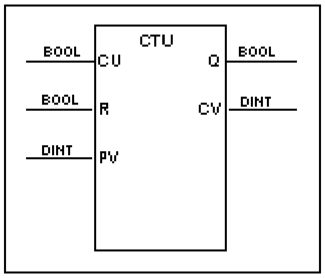
Other Counter Function Blocks

CTU	CTUD

CTU

The CTU function block counts recurring events up to a preset count, turning on the output when the current count is greater than or equal to the preset count.

Graphical representation:



Structured text syntax:

```
<CTU Name>(CU:=<count up>, R:=<reset>, PV:=<preset value>,
    EN:=<enable>, Q:=<output>, CV:=<current value>,
    ENO:=<enable output>,);
```

Parameter	Type	Description
CTU Name	CTU	Unique name for the counter.
CU	BOOL	Input increments the counter when CU transitions from FALSE to TRUE. Any BOOL constant or variable.
R	BOOL	Resets the counter. Any BOOL constant or variable.
PV	DINT	Contains the value up to which the CTU counts. Any DINT constant or variable.
EN	BOOL	Enables the counter. Any BOOL constant or variable.
Q	BOOL	Output changes to TRUE when CV = PV. Any BOOL constant or variable.
CV	DINT	Contains the current count of the counter. Any DINT variable.
ENO	BOOL	Echoes EN. Any BOOL constant or variable.

WARNING! Assigning the same function block name to different counters may cause unpredictable operation by the controller, which can result in death or injury to personnel and/or damage to equipment. Always use a unique name for each counter.

Operation is as follows:

- When the enable input EN is TRUE the counter is enabled.

  When EN is FALSE, the counter is not enabled and cannot begin counting.
- When reset R is set to TRUE, current value CV is set to zero, and output Q is set to FALSE.

  The counter does not begin counting while R is TRUE.
- The counter increments CV by one each time the count up input CU transitions from FALSE to TRUE.
- When CV = PV, the counter rung output Q is set to TRUE.
- If EN is set to FALSE while the counter is counting, the counter freezes Q and CV in their current states until EN is set to TRUE again.
- Enable output ENO echoes the value of EN.

When the program is running, you can double click on a CTU to display the CTU dialog box. The box displays the current value of all function block inputs and outputs. You can also open the watch window and enter counter variables that you want to observe at runtime.

You can use any of the CTU inputs and outputs in any expression, contact or coil instead of a symbol of the same type. To reference a CTU input or output, enter the function block name followed by a period and the specific input or output suffix. For example, CTU1.CU refers to the count up input of CTU1. All counter variables are listed in the following table.

Variable	**Reference Name**
CU	xxx.CU
R	xxx.R
PV	xxx.PV
EN	xxx.EN
Q	xxx.Q
CV	xxx.CV
ENO	xxx.ENO
Note xxx denotes the counter function block name.	

An example of the CTU timing diagram is shown in the figure at the end of this section. The sequence of events is listed below:

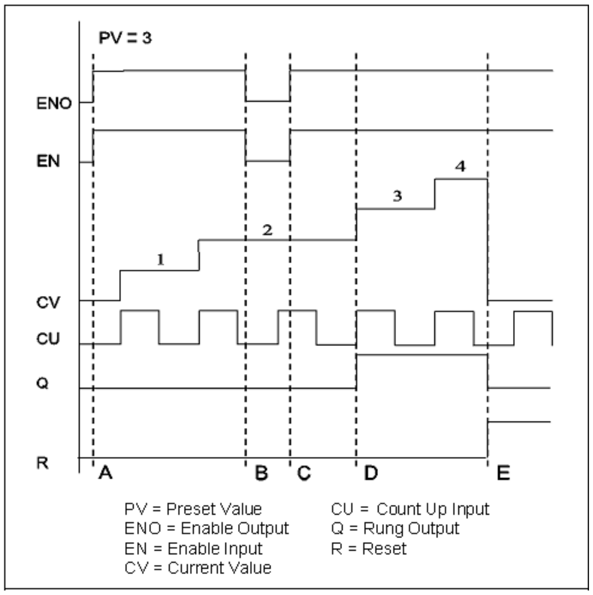
A. PV has been previously loaded with the preset value of 3.
EN and ENO transition from FALSE to TRUE.
CV contains the current value of 0.

B. EN transitions from TRUE to FALSE, disabling the counter.
CV, which had been counting up as CU changed state, holds at 2.

C. EN transitions from FALSE to TRUE, re-enabling the counter.
CV resumes counting up at the next FALSE-to-TRUE transition of CU, and reaches 3.
CV continues counting up until reset by R.

D. Q transitions to TRUE when CV=3.

E. R transitions from FALSE to TRUE, which resets CV to 0. CV holds at 0 while R is TRUE.



CTU Timing Diagram

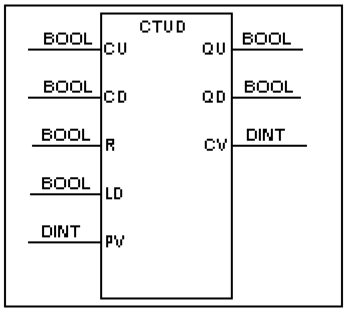
Other Counter Function Blocks

CTU	CTUD

CTUD

The CTUD function block counts recurring events up or down, turning on an up-count output when the current count is greater than or equal to the preset count, or a down-count output when the current count is less than or equal to zero.

Graphical representation:



Structured text syntax:

```
<CTUD Name>(CU:=<count up>, CD:=<count down>, R:=<reset>,
   LD:=<load>, PV:=<preset value>, EN:=<enable>,
   QU:=<count up output>, QD:=<count down output>,
   CV:=<current value>, ENO:=<enable output>,);
```

Parameter	Type	Description
CTUD Name	CTUD	Unique name for the counter.
CU	BOOL	Input increments current value CV when CU transitions from FALSE to TRUE. Any BOOL constant or variable.
CD	BOOL	Decrements current value CV when CD transitions from FALSE to TRUE. Any BOOL constant or variable.
R	BOOL	Sets current value CV to zero and sets the count-up output QU to FALSE. Any BOOL constant or variable.
LD	BOOL	Sets the current count CV to preset value PV and sets the count-down output to FALSE. Any BOOL constant or variable.
PV	DINT	Contains value to which CTUD counts up, or at which CTUD begins to count down. Any DINT constant or variable.
EN	BOOL	Enables the counter. Any BOOL constant or variable.
QU	BOOL	Output changes to TRUE when current value Output CV = preset value PV. Any BOOL constant or variable.
QD	BOOL	Changes to TRUE when current value CV = zero. QD is FALSE when CV > 0. Any BOOL constant or variable.

Parameter	Type	Description
CV	DINT	Contains the current count CV of the counter. Any DINT variable.
ENO	BOOL	Echoes EN. Valid values: any BOOL variable. Any BOOL constant or variable.

WARNING! Assigning the same function block name to different counters may cause unpredictable operation by the controller, which can result in death or injury to personnel and/or damage to equipment. Always use a unique name for each counter.

Counting Up or Down

Operation is as follows:

- When the enable input EN is TRUE the counter is enabled for up or down counting.

  When EN is FALSE, the counter is not enabled and cannot count up or down.
- If EN is set to FALSE while the counter is counting, current value CV, count-up output QU, and count-down output QD are frozen in their current states until EN is set to TRUE again.
- Enable output ENO echoes the value of EN.
- When reset R is set to TRUE, the load LD is disabled.

Counting Up

Operation is as follows:

- When reset R is set to TRUE, current value CV is set to zero and count-up output QU is set to FALSE.

  When R is TRUE, the counter cannot count, either up or down.
- The counter increments CV by one each time the count-up input CU transitions from FALSE to TRUE.
- When CV = the preset value PV, the counter sets QU to TRUE.

  QU is FALSE when CV < PV.

Counting Down

Operation is as follows:

- When load LD is set to TRUE, current value CV is set equal to PV and countdown output QD is set to FALSE.

  When LD is TRUE, the counter cannot count, either down or up.
- The counter decrements CV by one each time the count-down input CD transitions from FALSE to TRUE.

- When CV = zero, the counter sets QD to TRUE.

  QD is FALSE when CV > zero

When the program is running, you can double click on a CTUD to display the CTUD dialog box. The box displays the current value of all function block inputs and outputs. You can also open a watch window and enter counter variables that you want to observe at runtime.

You can use any of the CTUD inputs and outputs in any expression, contact or coil instead of a symbol of the same type. To reference a CTUD input or output, enter the function block name followed by a period and the specific input or output suffix. For example, CTUD1.CU refers to the count up input of CTUD1. All counter variables are listed in the following table.

Variable	**Reference Name**
CU	xxx.CU
CD	xxx.CD
R	xxx.R
LD	xxx.LD
PV	xxx.PV
EN	xxx.EN
QU	xxx.QU
QD	xxx.QD
CV	xx.CV
ENO	xxx.ENO
Note xxx denotes the counter function block name.	

An example of the CTUD timing diagram is shown in the figure at the end of this section. The sequence of events is listed below:

A. PV has been previously loaded with the preset value of 2.
EN and ENO transition from FALSE to TRUE.
QU is FALSE and QD is TRUE because CV contains the current value of 0.

B. CV, which had been counting up as CU changed state, reaches 2.
QU transitions to TRUE.

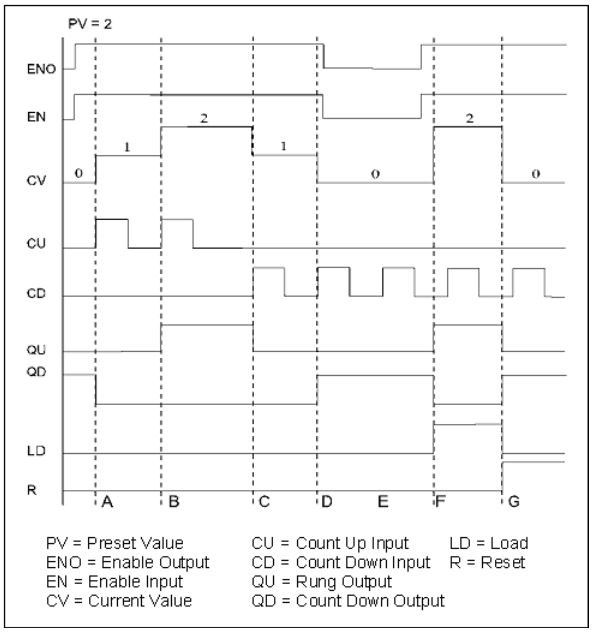
C. CV decrements to 1 when CD transitions from FALSE to TRUE.
QU transitions to FALSE.

D. CV decrements to 0 at the next transition of CD from FALSE to TRUE.
QD transitions to TRUE.

E. After point D, EN transitions from TRUE to FALSE, disabling the counter.
CV, QU, and QD are frozen in their current states. Prior to point E, EN transitions from FALSE to TRUE, re-enabling the counter.

F. LD transitions from FALSE to TRUE.
The preset value of 2 is loaded to CV, QD is set to FALSE, and QU is set to TRUE.

G. R transitions from FALSE to TRUE.
CV is set to 0, QU is set to FALSE, and QD is set to TRUE.



CTUD Timing Diagram

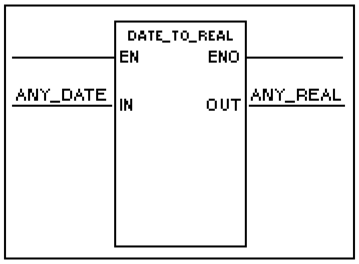
Other Counter Function Blocks

CTD	CTU

DATE_TO_REAL

The DATE_TO_REAL function converts a DATE data type input to an ANY_REAL value.

Graphical representation:



Structured text syntax:

```
<OUT> := DATE_TO_REAL (<IN>);
```

Parameter	Type	Description
IN	ANY_DATE	Contains the date value to convert. Any value, variable, or expression that resolves to an ANY_DATE data type (DT, DATE, TOD).
OUT	ANY_REAL	Contains the result of the conversion. An ANY_REAL data type (REAL, LREAL).

Operation is as follows:

- DATE_TO_REAL converts the value represented by the input variable and stores the result as an ANY_REAL data type in the output variable.

  The whole number portion of the real number represents the number of days since December 30, 1899. The fractional part of the number represents time of day.

- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Note To help ensure accuracy, use an LREAL data type for the result when converting a DT data type.

Examples of the DATE_TO_REAL operation:

DATE_TO_REAL(DT#2000-12-25:06:00:00) returns 36885.25

DATE_TO_REAL(TOD#18:00:00) returns 0.75

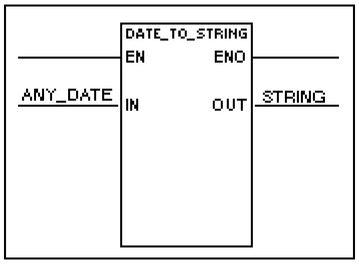
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_STRING
INT_TO_BCD	INT_TO_REAL	INT_TO_STRING
REAL_TO_DATE	REAL_TO_INT	REAL_TO_STRING
REAL_TO_TIME	STRING_TO_ARRAY	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

DATE_TO_STRING

The DATE_TO_STRING function converts a DATE data type input to a string.

Graphical representation:



Structured text syntax:

```
<OUT> := DATE_TO_STRING (<IN>);
```

Parameter	Type	Description
IN	ANY_DATE	Contains the date value to convert. Any value, variable, or expression that resolves to an ANY_DATE data type (DT, DATE, TOD).
OUT	STRING	Contains the result of the conversion. A variable (STRING data type).

Operation is as follows:

- DATE_TO_STRING converts the value represented by the input variable and stores the result as a STRING data type in the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the DATE_TO_STRING operation:

DATE_TO_STRING(DT#2000-12-25:06:00:00) returns the string 'DT#2000-12-25-06:00:00'

DATE_TO_STRING(DATE#2000-12-25) returns the string 'DATE#2000-12-25'

DATE_TO_STRING(TOD#18:00:00) returns the string 'TOD#18:00:00'

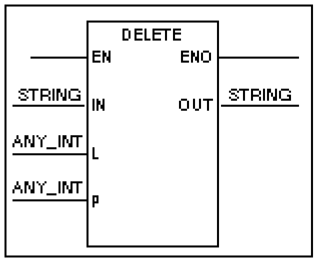
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
INT_TO_BCD	INT_TO_REAL	INT_TO_STRING
REAL_TO_DATE	REAL_TO_INT	REAL_TO_STRING
REAL_TO_TIME	STRING_TO_ARRAY	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

DELETE

The DELETE function deletes characters from the middle of an input string and stores the result into an output string.

Graphical representation:



Structured text syntax:

```
<OUT> := DELETE (IN:= <IN>, L:= <L>, P:= <P>);
```

Parameter	Type	Description
IN	STRING	Contains the string with characters to be deleted. Any valid STRING character or STRING variable.
L	ANY_INT	Specifies the number of characters to delete. Any SINT, INT, DINT, BYTE, WORD, DWORD constant or variable.
P	ANY_INT	Specifies the position within the string to begin deleting characters. For the first character in IN, P = 1. Any SINT, INT, DINT, BYTE, WORD, DWORD constant or variable.
OUT	STRING	Contains the string after the characters have been deleted. Any STRING variable.

Operation is as follows:

- DELETE deletes the characters, beginning at position P, up to L characters.
- DELETE stores the resulting string to output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function. Possible errors include a negative value in the Length field.

Example of the DEL operation:

DELETE('CDBROWN', 2, 4) returns 'CDBWN'

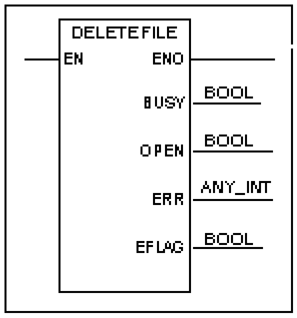
Other String Function Blocks

CONCAT	FIND	INSERT
LEFT	LEN	MID
MSGWND	REPLACE	RIGHT

DELETEFILE

The DELETEFILE function deletes a file. The DELETEFILE is one of eight functions that do file operations. Note that these function blocks are not designed for high-speed I/O execution or data transfers of large blocks of information.

Graphical representation:



Structured text syntax:

```
DELETEFILE (FCB:= <file control block name>,IN:=
   <filename>);
```

Parameter	Type	Description
File Control Block Name	FILE	Name of the file control block that handles operations for this file.
File Name	STRING	Name of the file to be deleted. The default path is the same as for the runtime engine. Any STRING constant or variable.
BUSY [1]	BOOL	Indicates the file control block is busy. Any BOOL constant or variable.
OPEN [1]	BOOL	Indicates the file has been opened. Any BOOL constant or variable.
EFLAG [1]	BOOL	Indicates a file operation error has occurred. Any BOOL constant or variable.
ERR [1]	ANY_INT	Contains error code if a file operation error occurs. Any SINT, INT, DINT, BYTE, WORD, DWORD variable.
1 Entries in the output fields are optional. However, for each field there is a default file control variable. As you design your program, you must use these output variables to handle file control. These fields reflect outputs in the specified file control block and are not actual parameters to the DELETEFILE function. For a detailed description of the file control variables, see "CLOSEFILE."		

Operation is as follows.

- DELETEFILE deletes the file that you specify in the File Name field.
- The file control variables handle access to the file and error conditions, as described for the "CLOSEFILE" function. All file functions that operate on the same file must use the same File Control Block name.
- If an error occurs, the file error variable is set to TRUE and a message appears in the Output window and the Wonderware Logger. Attempting to delete an open file generates an error (error code 27). For a complete list of the error codes, the "CLOSEFILE" function.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

You can do only one file operation for each File Control Block at a time. Note that file control I/O operations take place asynchronously to program execution.

Example of the DELETEFILE function:

```
DELETEFILE (FCB:= datrpt IN:= "datareport");
```

The system deletes the file called datareport.

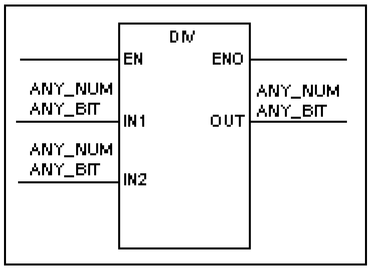
Other File Function Blocks

CLOSEFILE	COPYFILE	DIV
NEWFILE	OPENFILE	READFILE
REWINDFILE	WRITEFILE	

DIV

The DIV function divides one input value by another.

Graphical representation:



Structured text syntax:

```
<OUT> := <IN1> / <IN2>;
```

Parameter	Type	Description
IN1	ANY_NUM ANY_BIT [1]	Contains the dividend. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
IN2	ANY_NUM ANY_BIT [1]	Contains the divisor. Any non-zero SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
OUT	ANY_NUM ANY_BIT [1]	Contains the quotient of the division of IN1 by IN2. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD variable.
1 BOOL data types are not allowed.		

Operation is as follows:

- DIV divides IN1 by IN2, stores the quotient to the output variable OUT.
- If the value of IN2 equals zero, IN2 is set to 1 and the runtime engine system variable RTEngine.DivideZero is set to TRUE.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

You can also use this function with arrays. For example, you can divide every element in an array by a number with the following line:

Array1 := Array2 / 9;

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

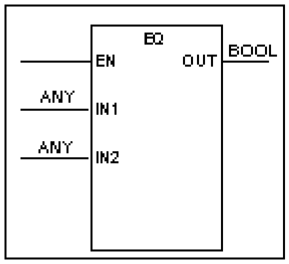
Other Math Functions

ABS	ADD	EXPT
MAX	MIN	MOD
MOVE	MUL	NEG
SQRT	SUB	TRUNC

EQ

The EQ function block tests whether one input is equal to another input.

Graphical representation:



Structured text syntax:

```
<OUT> := (<IN1> = <IN2>);
```

Parameter	Type	Description
IN1	ANY	Contains first value to be compared. Any data type.
IN2	ANY	Contains second value to be compared. Any data type.
OUT	BOOL	TRUE indicates the values are equal. FALSE indicates the values are not equal. Any BOOL variable.

Operation is as follows:

- EQ compares IN1 to IN2. If IN1 is equal to IN2, EQ sets the output variable OUT to TRUE. Otherwise, EQ sets OUT to FALSE.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute.

Note In general, it is recommended that you avoid doing a comparison for equality (or non-equality) with real numbers. If you do this type of comparison using a constant (literal) value and a real variable, the variable must be an LREAL data type to help ensure that you receive the expected result.

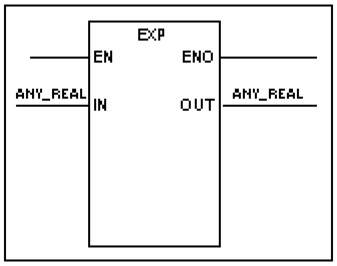
Other Comparison Function Blocks

GE	GT	LE
LT	NE	

EXP

The EXP function block calculates the natural log exponentiation of the input (raises e to the power of the input).

Graphical representation:



Structured text syntax:

```
<OUT> := EXP (<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the value used as the exponent for e. Any REAL or LREAL constant or variable.
OUT	ANY_REAL	Contains the result of e raised to the power of IN. Any REAL or LREAL variable.

Operation is as follows:

- If the input is within the range of the selected data type, EXP calculates e to the power of IN, stores the result to the output variable OUT.
- If an error occurs, zero is written to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the EXP function:

EXP(1) returns the value of e (2.7128...)

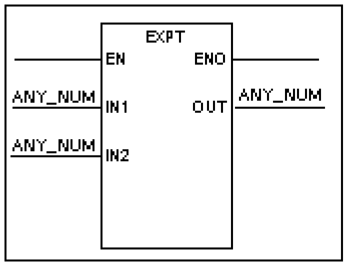
Other Trig/Log Function Blocks

ACOS	ASIN	ATAN
COS	LN	LOG
SIN	TAN	

EXPT

The EXPT function block raises a value to the power specified by a second value.

Graphical representation:



Structured text syntax:

```
<OUT> := (<IN1> ** <IN2>);
```

Parameter	Type	Description
IN1	ANY_NUM	Contains the value to be raised to the power of IN2. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
IN2	ANY_NUM	Contains the value used as the exponent for IN1. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
OUT	ANY_NUM	Contains the result of IN1 raised to the power of IN2. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD variable.

Operation is as follows:

- EXPT calculates IN1 raised to the power of IN2, stores the result to the output variable OUT.
- For the specific case in which both IN1 and IN2 equal zero, OUT equals one.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.
- If an error occurs, zero is written to the output variable OUT.

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

Examples of the EXPT operation:

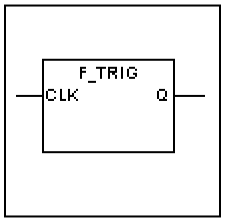
Other Math Functions

ABS	ADD	DIV
MAX	MIN	MOD
MOVE	MUL	NEG
SQRT	SUB	TRUNC

F_TRIG

The F_TRIG function block sets an output to TRUE for one scan when the input to the function block transitions from TRUE to FALSE.

Graphical representation:



Structured text syntax:

<F_TRIG Name> `(CLK:=` **<CLK>**`, Q:=` **<Q>**`);`

Parameter	Type	Description
F_TRIG Name	F_TRIG	Unique name for the function block.
CLK	BOOL	Enables the function block output Q when CLK transitions from TRUE to FALSE. Any BOOL variable (or rung input).
Q	BOOL	Output is set to TRUE when CLK transitions from TRUE to FALSE. Any BOOL variable (or rung output).

Operation is as follows:

- When the input to the F_TRIG, which is from the clock, transitions from TRUE to FALSE, the F_TRIG sets output Q to TRUE for one scan.

You can use the F_TRIG input and output in any expression, contact or coil instead of a symbol of the same type. To reference an F_TRIG input or output, enter the function block name followed by a period and the specific input or output suffix. For example, F_TRIG1.CLK refers to the clock input of F_TRIG1.

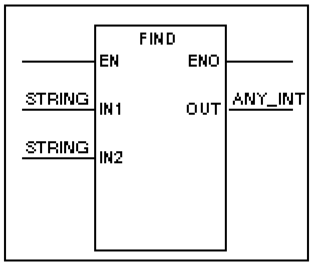
Other Trigger Function Blocks

ABORT_ALL	R_TRIG

FIND

The FIND function block searches for a string of characters within another string.

Graphical representation:



Structured text syntax:

```
<OUT> := FIND (IN1:= <IN1>, IN2:= <IN2>);
```

Parameter	Type	Description
IN1	STRING	Contains the string of characters to be searched. Any valid STRING character or variable.
IN2	STRING	Contains the string of characters for which a match is to be found. Any valid STRING character or variable.
OUT	ANY_INT	Contains the result of the search. Any SINT, INT, DINT, BYTE, WORD, DWORD variable.

Operation is as follows:

- FIND searches string IN1 for a match of the string of characters in string IN2. FIND is case sensitive.
- If a match is found, FIND stores the starting character position of IN2 within IN1 to output variable OUT. The first character position is number 1.
- If IN2 is not found, zero is stored to OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Examples of the FIND operation:

FIND('CDBROWN', 'BR') returns 3

FIND('CDBROWN', 'SMITH') returns 0

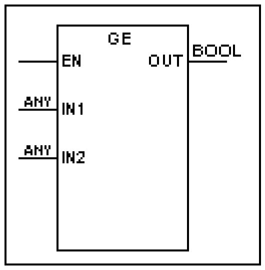
Other String Function Blocks

CONCAT	DELETE	INSERT
LEFT	LEN	MID
MSGWND	REPLACE	RIGHT

GE

The GE function tests whether one input is greater than or equal to another input.

Graphical representation:



Structured text syntax:

```
<OUT> := (<IN1> >= <IN2>);
```

Parameter	Type	Description
IN1	ANY	Contains first value to be compared. Any data type.
IN2	ANY	Contains second value to be compared. Any data type.
OUT	BOOL	TRUE indicates IN1 is greater than or equal to than IN2. FALSE indicates IN1 is less than IN2. Any BOOL variable.

Operation is as follows:

- GE compares IN1 to IN2. If IN1 is greater than or equal to IN2, GE sets the output variable OUT to TRUE. Otherwise, the system sets OUT to FALSE.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute.

Note In general, it is recommended that you avoid doing a comparison for equality (or non-equality) with real numbers. If you do this type of comparison using a constant (literal) value and a real variable, the variable must be an LREAL data type to help ensure that you receive the expected result.

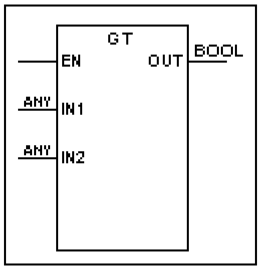
Other Comparison Functions

EQ	GT	LE
LT	NE	

GT

The GT function block tests whether one input is greater than another input.

Graphical representation:



Structured text syntax:

```
<OUT> := (<IN1> > <IN2>);
```

Parameter	Type	Description
IN1	ANY	Contains first value to be compared. Any data type.
IN2	ANY	Contains second value to be compared. Any data type.
OUT	BOOL	TRUE indicates IN1 is greater than IN2. FALSE indicates IN1 is less than or equal to IN2. Any BOOL variable.

Operation is as follows:

- The system compares IN1 to IN2. If IN1 is greater than IN2, GT sets the rung output variable OUT to TRUE. Otherwise, GE sets OUT to FALSE.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute.

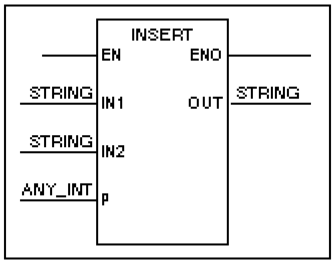
Other Comparison Functions

EQ	GE	LE
LT	NE	

INSERT

The INS function block inserts a string input into another string.

Graphical representation:



Structured text syntax:

```
<OUT> := INSERT (IN1:= <IN1>, IN2:= <IN2> , P:= <P>);
```

Parameter	Type	Description
IN1	STRING	Contains the string of characters into which another string is to be inserted. Any valid STRING character or variable.
IN2	STRING	Contains the string of characters that is to be inserted into IN1. Any STRING character or variable.
P	ANY_INT	Specifies the character position of IN1 after which IN2 is inserted. To insert IN2 before the first character in IN1, set P = 0. Any SINT, INT, DINT, BYTE, WORD, DWORD constant or variable.
OUT	STRING	Contains the result of the string insertion. Any valid STRING variable.

Operation is as follows:

- INSERT inserts string IN2 into string IN1 at the position specified by position P.
- INSERT stores the result of the insertion into output variable OUT, and sets the ENO to TRUE.
- If the value of P is negative, or if the sum of the lengths of the two strings is greater than 2048, OUT is set to an empty string. If P is greater than the length of the string, the strings are concatenated.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the INS operation when P = 4:

INSERT('BROWN', 'CD', 4) returns 'BROWCDN'

Example of the INS operation when P = 0:

INSERT('BROWN', 'CD', 0) returns 'CDBROWN'

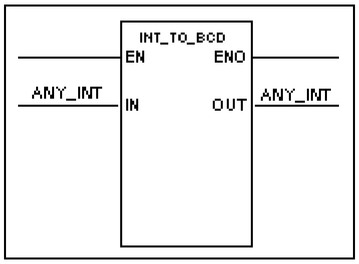
Other String Functions

CONCAT	DELETE	FIND
LEFT	LEN	MID
MSGWND	REPLACE	RIGHT

INT_TO_BCD

The INT_TO_BCD function converts an integer value to the equivalent Binary-Coded Decimal (BCD) representation of the value.

Graphical representation:



Structured text syntax:

```
<OUT> := INT_TO_BCD (<IN>);
```

Parameter	Type	Description
IN	ANY_INT	Contains the value to convert. An ANY_INT variable data type (SINT, INT, DINT, BYTE, WORD, DWORD).
OUT	ANY_INT	Contains the result of the conversion. A BCD number, variable or expression that resolves to an ANY_INTdata type (SINT, INT, DINT, BYTE, WORD, DWORD).

Operation is as follows:

- INT_TO_BCD converts the integer input to its BCD representation.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the INT_TO_BCD function.

INT_TO_BCD(10#321) returns 16#321

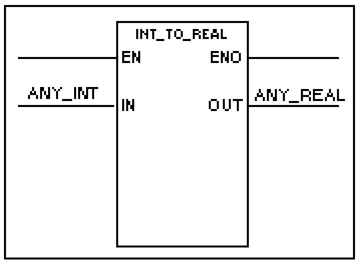
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_REAL	INT_TO_STRING
REAL_TO_DATE	REAL_TO_INT	REAL_TO_STRING
REAL_TO_TIME	STRING_TO_ARRAY	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

INT_TO_REAL

The INT_TO_REAL function converts an ANY_INT input to an ANY_REAL value.

Graphical representation:



Structured text syntax:

```
<OUT> := INT_TO_REAL (<IN>);
```

Parameter	Type	Description
IN	ANY_INT	Contains the value to convert. An ANY_INT variable data type (SINT, INT, DINT, BYTE, WORD, DWORD).
OUT	ANY_REAL	Contains the result of the conversion. An ANY_REAL data type (REAL, LREAL).

Operation is as follows:

- INT_TO_REAL converts the integer input and stores it as an LREAL data type in the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the INT_TO_REAL function.

INT_TO_REAL(4) returns 4.0.

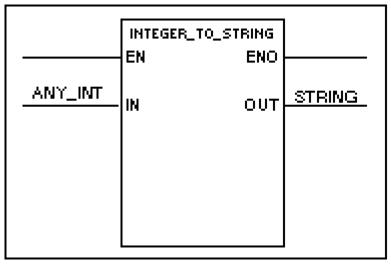
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_STRING
REAL_TO_DATE	REAL_TO_INT	REAL_TO_STRING
REAL_TO_TIME	STRING_TO_ARRAY	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

INT_TO_STRING

The INT_TO_STRING function converts an ANY_INT input to a string.

Graphical representation:



Structured text syntax:

```
<OUT> := INT_TO_STRING (<IN>);
```

Parameter	Type	Description
IN	ANY_INT	Contains the value to convert. An ANY_INT variable data type (SINT, INT, DINT, BYTE, WORD, DWORD).
OUT	STRING	Contains the result of the conversion. A variable (STRING data type). Any BOOL variable.

Operation is as follows:

- INT_TO_STRING converts the integer input and stores it as a STRING data type in the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the INT_TO_STRING function.

INT_TO_STRING(4) returns the string '4'

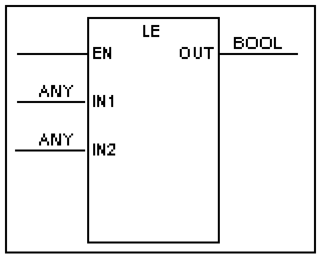
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
REAL_TO_DATE	REAL_TO_INT	REAL_TO_STRING
REAL_TO_TIME	STRING_TO_ARRAY	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

LE

The LE function tests whether one input is less than another input.

Graphical representation:



Structured text syntax:

```
<OUT>:=(<IN1> <= <IN2>);
```

Parameter	Type	Description
IN1	ANY	Contains first value to be compared. Any data type.
IN2	ANY	Contains second value to be compared. Any data type.
OUT	BOOL	TRUE indicates IN1 is less than or equal to than IN2. FALSE indicates IN1 is greater than IN2. Any BOOL variable.

Operation is as follows:

- LE compares IN1 to IN2. If IN1 is less than or equal to IN2, LE sets the output variable OUT to TRUE. Otherwise, LE sets OUT to FALSE.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute.

Note In general, it is recommended that you avoid doing a comparison for equality (or non-equality) with real numbers. If you do this type of comparison using a constant (literal) value and a real variable, the variable must be an LREAL data type to help ensure that you receive the expected result.

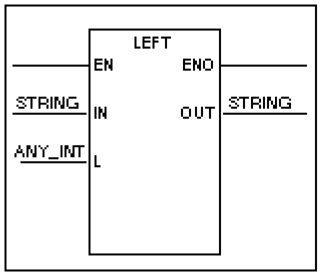
Other Comparison Functions

EQ	GE	GT
LT	NE	

LEFT

The LEFT function creates a string of characters from a specified number of the leftmost characters of another string of characters.

Graphical representation:



Structured text syntax:

```
<OUT> := LEFT (IN:= <IN> , L:= <L>);
```

Parameter	**Type**	**Description**
IN	STRING	Contains string from which characters are copied. Any valid STRING character or variable.
L	ANY_INT	Specifies the number of characters to copy. Any SINT, INT, DINT, BYTE, WORD, DWORD constant or variable.
OUT	STRING	Contains the new string of characters. Any valid STRING variable.

Operation is as follows:

- LEFT copies the number of characters specified by L, starting from the left end of a string specified by IN, to the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function. Possible errors include a negative value in the L field.

Example of the LEFT operation:

LEFT('BROWN', 2) returns 'BR'

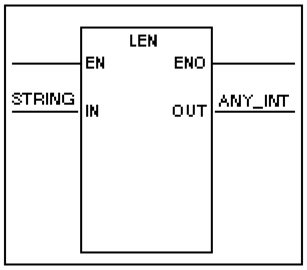
Other String Functions

CONCAT	DELETE	FIND
INSERT	LEN	MID
MSGWND	REPLACE	RIGHT

LEN

The LEN function stores the length of a string.

Graphical representation:



Structured text syntax:

```
<OUT> := LEN (<IN>);
```

Parameter	Type	Description
IN	STRING	Contains the string for which the length is stored. Any valid STRING character or variable.
OUT	ANY_INT	Contains the integer length of the string. Any valid SINT, INT, DINT, BYTE, WORD, DWORD variable.

Operation is as follows:

- LEN determines the length of the string specified in IN, and stores the result to the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

The following is an example of the LEN operation:

LEN('BROWN') returns 5

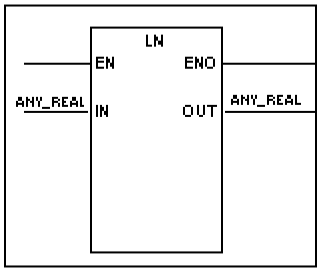
Other String Functions

CONCAT	DELETE	FIND
INSERT	LEFT	MID
MSGWND	REPLACE	RIGHT

LN

The LN function calculates the natural logarithm of the input.

Graphical representation:



Structured text syntax:

```
<OUT> := LN (<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the value for which the natural logarithm is calculated. Any REAL or LREAL constant or variable.
OUT	ANY_REAL	Contains the natural logarithm of IN. Any REAL or LREAL variable.

Operation is as follows:

- LN calculates the natural log of IN, stores the result to the output variable OUT, and sets the rung output ENO to TRUE.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the LN function:

LN(e) returns 1 where e is 2.7128...

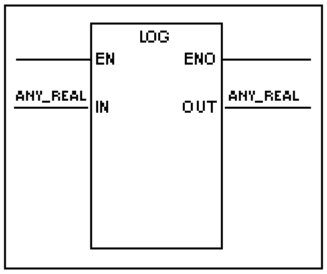
Other Trig/Log Functions

ACOS	ASIN	ATAN
COS	EXP	LOG
SIN	TAN	

LOG

The LOG function calculates the base 10 logarithm of the input.

Graphical representation:



Structured text syntax:

```
<OUT> := LOG (<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the value for which the logarithm is calculated. Any REAL or LREAL constant or variable.
OUT	ANY_REAL	Contains the logarithm of IN. Any REAL or LREAL variable.

Operation is as follows:

- LOG calculates the base 10 logarithm of IN, stores the result to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the LOG function:

LOG(10) returns 1

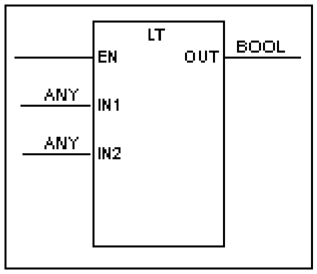
Other Trig/Log Functions

ACOS	ASIN	ATAN
COS	EXP	LN
SIN	TAN	

LT

The LT function tests whether one input is less than another input.

Graphical representation:



Structured text syntax:

```
<OUT> := (<IN1> < <IN2>);
```

Parameter	Type	Description
IN1	ANY	Contains first value to be compared. Any data type.
IN2	ANY	Contains second value to be compared. Any data type.
OUT	BOOL	TRUE indicates IN1 is less than IN2. FALSE indicates IN1 is greater than or equal to IN2. Any BOOL variable.

Operation is as follows: OUT := (IN1 < IN2)

- LT compares IN1 to IN2. If IN1 is less than IN2, LT sets the output variable OUT to TRUE. Otherwise, LT sets OUT to FALSE.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute.

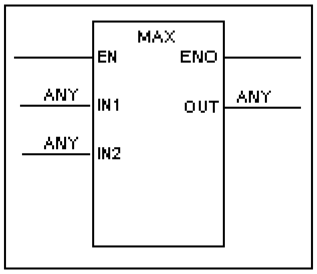
Other Comparison Functions

EQ	GE	GT
LE	NE	

MAX

The MAX function determines the larger of two values.

Graphical representation:



Structured text syntax:

```
<OUT> := MAX (<IN1> , <IN2>);
```

Parameter	Type	Description
IN1	ANY	Contains first value to be compared. Any numbers, variables, or expressions that resolve to an ANY data type except FILE, TMR, and User-Defined.
IN2	ANY	Contains second value to be compared. Any numbers, variables, or expressions that resolve to an ANY data type except FILE, TMR, and User-Defined.
OUT	ANY	Contains the larger value. An ANY variable data type except FILE, TMR, and User-Defined.

Operation is as follows:

- MAX compares IN1 to IN2 and stores the larger value in the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function. Possible errors include a negative value in the L field.

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

Example of the MAX function:

MAX(5.4, 9.0) returns 9.0

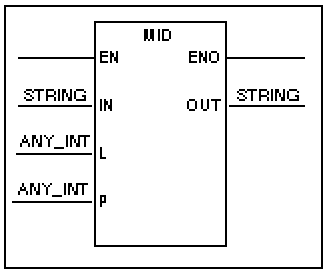
Other Math Functions

ABS	ADD	DIV
EXPT	MIN	MOD
MOVE	MUL	NEG
SQRT	SUB	TRUNC

MID

The MID function creates a string of characters from a specified number of characters from the middle of another string.

Graphical representation:



Structured text syntax

```
<OUT> := MID (IN:= <IN>, L:= <L> , P:= <P>);
```

Parameter	Type	Description
IN	STRING	Contains string from which characters are copied. Any valid STRING character or variable.
L	ANY_INT	Specifies the number of characters to copy. Any SINT, INT, DINT, BYTE, WORD, DWORD constant or variable.
P	ANY_INT	Specifies the position within the string to begin copying characters. Any SINT, INT, DINT, BYTE, WORD, DWORD constant or variable.
OUT	STRING	Contains the new string of characters. Any valid STRING variable.

Operation is as follows:

- MID copies the number of characters specified by L and beginning at position P to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function. Possible errors include a negative value in the L field or if the value for P falls outside the string.

Example of the MID operation:

MID('BROWN', 2, 4) returns 'WN'

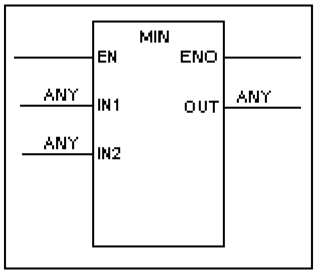
Other String Functions

CONCAT	DELETE	FIND
INSERT	LEFT	LEN
MSGWND	REPLACE	RIGHT

MIN

The MIN function determines the smaller of two values.

Graphical representation:



Structured text syntax:

```
<OUT> := MIN (<IN1> , <IN2>);
```

Parameter	Type	Description
IN1	ANY	Contains first value to be compared. Any numbers, variables, or expressions that resolve to an ANY data type except FILE, TMR, and User-Defined.
IN2	ANY	Contains second value to be compared. Any numbers, variables, or expressions that resolve to an ANY data type except FILE, TMR, and User-Defined.
OUT	ANY	Contains the smaller value. An ANY variable data type except FILE, TMR, and User-Defined.

Operation is as follows:

- MIN compares IN1 to IN2 and stores the smaller value in the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the MIN function:

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

MIN(5.4, 9.0) returns 5.4

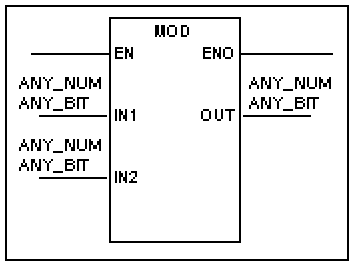
Other Math Functions

ABS	ADD	DIV
EXPT	MAX	MOD
MOVE	MUL	NEG
SQRT	SUB	TRUNC

MOD

The MOD function divides one input value by another and stores the remainder of the division (modulus) to the output.

Graphical representation:



Structured text syntax:

```
<OUT> := (<IN1> MOD <IN2>);
```

Parameter	Type	Description
IN1	ANY_NUM ANY_BIT [1]	Contains the dividend. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
IN2	ANY_NUM ANY_BIT [1]	Contains the divisor. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
OUT	ANY_NUM ANY_BIT [1]	Contains modulus of division of IN1 by IN2. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD variable.
1 BOOL data types are not allowed.		

Operation is as follows:

- If the two inputs IN1and IN2 are within the range of the selected data types, MOD divides IN1 by IN2, stores the modulus to the output variable OUT.
- If the value of IN2 equals zero, IN2 is set to 1 and the runtime engine system variable RTEngine.DivideZero is set to TRUE.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

Example of the MOD function:

MOD(42, 10) returns 2

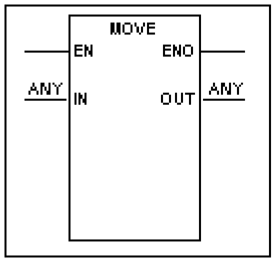
Other Math Functions

ABS	ADD	DIV
EXPT	MAX	MIN
MOVE	MUL	NEG
SQRT	SUB	TRUNC

MOVE

The MOVE function converts the input to the same data type as the output and copies the result to the output.

Graphical representation:



Structured text syntax:

```
<OUT> := <IN>);
```

Parameter	Type	Description
IN	ANY	Contains the value to be copied. Any data type.
OUT	ANY	Contains the destination of the copy operation. Any data type.

Operation is as follows:

- MOVE converts the input IN to the same data type as the output, copies the result to the output variable OUT.
- The contents of IN are not affected by the operation.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Note Moving structures and arrays is possible although this type of operation may be lengthy, depending on the size of the structure or array.

To move a structure (Structure1:= Structure2), the size and data types of the structure members must match exactly. No data type conversion is supported for complex moves.

To move an array (Array1:= Array2), the size and data types of the array elements must match exactly. Each element of Array2 is moved to a corresponding element in Array1.

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

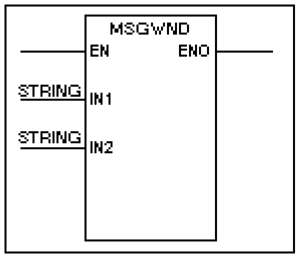
Other Math Functions

ABS	ADD	DIV
EXPT	MAX	MIN
MOD	MUL	NEG
SQRT	SUB	TRUNC

MSGWND

The MSGWND function block displays a message in the Output window and the Wonderware Logger.

Graphical representation:



Structured text syntax:

```
MSGWND (IN1:= <IN1> , IN2:= <IN2>);
```

Parameter	Type	Description
FBN	INT	Unique number for the function block.
IN1	STRING	Contains the message to be displayed, enclosed in single quotation marks. Any valid STRING character code or variable containing valid character codes.
IN2	STRING	Contains the title of the message, enclosed in single quotation marks. Any valid STRING character code or variable containing valid character codes.

Operation is as follows:

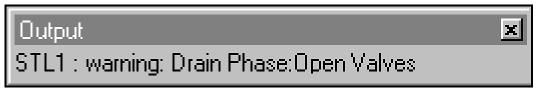
- MSGWND displays the contents of IN1 and IN2 in the Output window and the Wonderware Logger. The runtime engine monitor icon displays the yellow warning diamond:

- The EN BOOL parameter is used only in the graphical languages to enable the function block to execute. ENO follows EN unless an error condition occurs within the function block.

Example of the MSGWND operation:

```
MSGWND ('Open Valves', Phase);
```

The following figure shows the Output window when the Phase is 'Drain Phase.'



```
STL1 : warning: Drain Phase:Open Valves
```

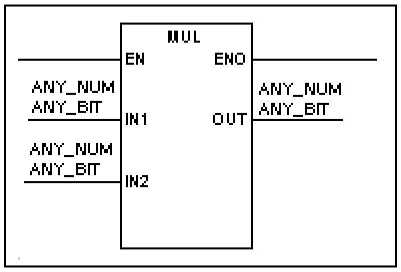
Other String Functions

CONCAT	DELETE	FIND
INSERT	LEFT	LEN
MID	REPLACE	RIGHT

MUL

The MUL function multiplies two input values.

Graphical representation:



Structured text syntax:

```
<OUT> := <IN1> * <IN2>
```

Parameter	Type	Description
IN1	ANY_NUM ANY_BIT [1]	Contains the first value to be multiplied. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
IN2	ANY_NUM ANY_BIT [1]	Contains the second value to be multiplied. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
OUT	ANY_NUM ANY_BIT [1]	Contains the product of the multiplication of IN1 and IN2. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD variable.
1 BOOL data types are not allowed.		

Operation is as follows:

- MUL multiplies IN1 and IN2 and stores the product to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

You can also use this function with arrays. For example, you can multiply every element in an array by a number with the following line:

Array1 := Array2 * 9;

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

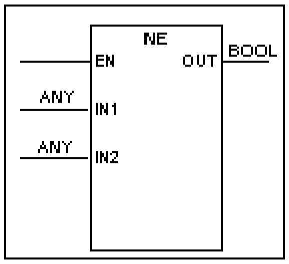
Other Math Functions

ABS	ADD	DIV
EXPT	MAX	MIN
MOD	MOVE	NEG
SQRT	SUB	TRUNC

NE

The NE function tests whether one input is not equal to another input.

Graphical representation:



Structured text syntax:

```
<OUT> := (<IN1> <> <IN2>);
```

Parameter	Type	Description
IN1	ANY	Contains first value to be compared. Any data type.
IN2	ANY	Contains second value to be compared. Any data type.
OUT	BOOL	TRUE indicates the values are not equal. FALSE indicates the values are equal. Any BOOL variable.

Operation is as follows:

- NE compares IN1 to IN2. If IN1 is not equal to IN2, NE sets output variable OUT to TRUE. Otherwise, NE sets OUT to FALSE.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute.

Note In general, it is recommended that you avoid doing a comparison for equality (or non-equality) with real numbers. If you do this type of comparison using a constant (literal) value and a real variable, the variable must be an LREAL data type to help ensure that you receive the expected result.

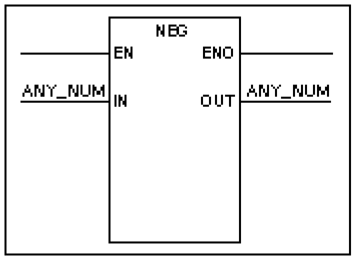
Other Comparison Function Blocks

EQ	GE	GT
LE	LT	

NEG

The NEG function changes the sign of an input.

Graphical representation:



Structured text syntax:

```
<OUT> := -<IN1>);
```

Parameter	Type	Description
IN	ANY_NUM	Contains the value to be negated. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
OUT	ANY_NUM	Contains the negated value of IN. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD variable.

Operation is as follows:

- NEG changes the sign of input IN and stores the result to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

Examples of the NEG operation:

NEG(-4) returns 4

NEG(4) returns -4

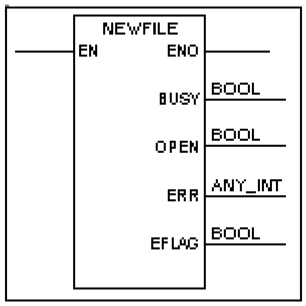
Other Math Functions

ABS	ADD	DIV
EXPT	MAX	MIN
MOD	MOVE	MUL
SQRT	SUB	TRUNC

NEWFILE

The NEWFILE function creates a new file. The NEWFILE is one of eight functions that do file operations. Note that these functions are not designed for high-speed I/O execution or data transfers of large blocks of information.

Graphical representation:



Structured text syntax:

```
NEWFILE (FCB:= <file control block name>, FILE:=
    <filename>);
```

Parameter	Type	Description
File Control Block Name	FILE	Name of the file control block that handles operations for this file.
File Name	STRING	Name of the file to be created. The default path is the same as for the runtime engine (RTEngine.exe). If you need to specify a different path, do not use a UNC (Universal Naming Convention) name. UNC is not supported. Any valid STRING constant or variable.
BUSY [1]	BOOL	Indicates the file control block is busy. Any BOOL variable.
OPEN [1]	BOOL	Indicates the file has been opened. Any BOOL variable.
EFLAG [1]	BOOL	Indicates a file operation error has occurred. Any BOOL variable.
ERR [1]	ANY_INT	Contains the error code if a file operation error occurs. Any SINT, INT, DINT, BYTE, WORD, DWORD variable.
1 Entries in the output fields are optional. However, for each field there is a default file control variable. As you design your program, you must use these output variables to handle file control. These fields reflect outputs in the specified file control block and are not actual parameters to the NEWFILE function. For a detailed description of the file control variables, see "CLOSEFILE."		

Operation is as follows.

- NEWFILE creates the file and assigns it the name that you specify in the File Name field. If a file of the same name already exists, NEWFILE overwrites it. If a file being handled by the file control block is already open, NEWFILE closes the first file and creates the second.
- The file control variables handle access to the file and error conditions, as described for the "CLOSEFILE" function. The three input file control variables are of particular importance when files are opened. You need to verify whether their default values are appropriate for your application. All file function blocks that operate on the same file must use the same File Control Block name.
- If an error occurs, the file error variable is set to TRUE and a message appears in the Output window and the Wonderware Logger. The graphical output ENO is set to FALSE. For a complete list of the error codes, see "CLOSEFILE."

You can do only one file operation for each File Control Block at a time. Note that file control I/O operations take place asynchronously to program execution.

For an example that uses the NEWFILE with several other File procedures, see "WRITEFILE."

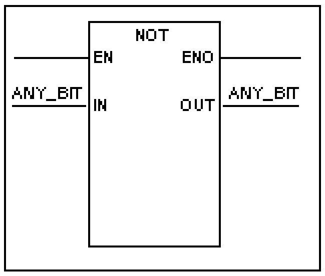
Other File Functions

CLOSEFILE	COPYFILE	DELETEFILE
OPENFILE	READFILE	REWINDFILE
WRITEFILE		

NOT

The NOT function does a bit-by-bit inversion of an input.

Graphical representation:



Structured text syntax:

```
<OUT> := NOT <IN>;
```

Parameter	Type	Description
IN	ANY_BIT	Contains the value to be inverted. Any BOOL, BYTE, WORD, DWORD constant or variable.
OUT	ANY_BIT	Contains the inverted value of IN. Any BOOL, BYTE, WORD, DWORD constant or variable.

Operation is as follows:

- NOT examines input IN bit by bit and inverts each bit.
- NOT stores the result to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

You can also use this function with arrays as shown in the example below.

Examples of the NOT operation:

NOT 2#0011 returns 2#1100

NOT TRUE returns FALSE

Array1 := NOT Array2;

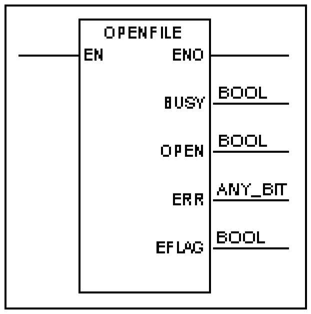
Other Bitwise Functions

AND	OR	ROL
ROR	SHL	SHR
XOR		

OPENFILE

The OPENFILE function opens a file for operations, such as reading or writing. The OPENFILE is one of eight function blocks that do file operations. Note that these function blocks are not designed for high-speed I/O execution or data transfers of large blocks of information.

Graphical representation:



Structured text syntax:

```
OPENFILE (FCB:= <file control block name>, FILE:=
    <filename>);
```

Parameter	Type	Description
File Control Block Name	FILE	Name of the file control block that handles operations for this file.
File Name	STRING	Name of the file to be opened. The default path is the same as for the runtime engine (RTEngine.exe). If you need to specify a different path, do not use a UNC (Universal Naming Convention) name. UNC is not supported. Any valid STRING constant or variable.
BUSY [1]	BOOL	Indicates the file control block is busy. Any BOOL variable.
OPEN [1]	BOOL	Indicates the file has been opened. Any BOOL variable.
EFLAG [1]	BOOL	Indicates a file operation error has occurred. Any BOOL variable.
ERR [1]	ANY_INT	Contains the error code if a file operation error occurs. Any SINT, INT, DINT, BYTE, WORD, DWORD variable.
1	Entries in the output fields are optional. However, for each field there is a default file control variable. As you design your program, you must use these output variables to handle file control. These fields reflect outputs in the specified file control block and are not actual parameters to the OPENFILE function. For a detailed description of the file control variables, see "CLOSEFILE."	

Operation is as follows.

- OPENFILE opens the file that you specify in the File Name field. If the file is already open, OPENFILE closes it first.
- The file control variables handle access to the file and error conditions, as described for the "CLOSEFILE" function block. The three input file control variables are of particular importance when files are opened. You need to verify whether their default values are appropriate for your application. All file function blocks that operate on the same file must use the same File Control Block name.
- If an error occurs, the file error variable is set to TRUE and a message appears in the Output window and the Wonderware Logger. The graphical output ENO is set to FALSE. Attempting to open a non-existent file generates an error (error code 18). If a file being handled by the file control block is already open, OPENFILE closes the first and opens the second. For a complete list of the error codes, see "CLOSEFILE."

You can do only one file operation for each File Control Block at a time. Note that file control I/O operations take place asynchronously to program execution.

The following is an example of the OPENFILE procedure.

```
OPENFILE (FCB:= "datrpt", FILE:= "data_report");
```

The system opens the file called data_report.

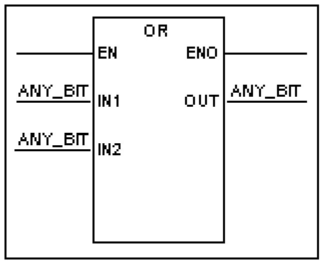
Other File Functions

CLOSEFILE	COPYFILE	DELETEFILE
NEWFILE	READFILE	REWINDFILE
WRITEFILE		

OR

The OR function does a bitwise logical OR of two values.

Graphical representation:



Structured text syntax:

```
<OUT> := <IN1> OR <IN2>;
```

Parameter	Type	Description
IN1	ANY_BIT	Contains the first value to be ORed. Any BOOL, BYTE, WORD, DWORD constant or variable.
IN2	ANY_BIT	Contains the second value to be ORed. Any BOOL, BYTE, WORD, DWORD constant or variable.

Operation is as follows:

- OR does a logical OR of each bit of the two inputs IN1and IN2.
- OR stores the result of the OR operation to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

You can also use this function with arrays as shown in the examples below.

Examples of the OR operation:

2#0011 OR 2#0101 returns 2#0111

TRUE OR FALSE returns TRUE

Array1 := Array2 OR #16FFFE;

Array1 := Array1 OR Array2;

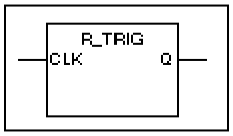
Other Bitwise Functions

AND	NOT	ROL
ROR	SHL	SHR
XOR		

R_TRIG

The R_TRIG function block sets an output to TRUE when the input to the function block transitions from FALSE to TRUE.

Graphical representation:



Structured text syntax:

```
<R_TRIG Name> (CLK:= <CLK>, Q:= <Q>);
```

Parameter	Type	Description
R_TRIG Name	R_TRIG	Unique name for the function block.
CLK	BOOL	Enables the function block output Q when CLK transitions from FALSE to TRUE. Any BOOL variable (or rung input).
Q	BOOL	Output is set to TRUE when CLK transitions from FALSE to TRUE. Any BOOL variable (or rung input).

Operation is as follows.

- When the input to the R_TRIG, which is from the clock, transitions from FALSE to TRUE, the R_TRIG sets output Q to TRUE for one scan.

You can use the R_TRIG input and output in any expression, contact or coil instead of a symbol of the same type. To reference an R_TRIG input or output, enter the function block name followed by a period and the specific input or output suffix. For example, R_TRIG1.Q refers to the output of R_TRIG1.

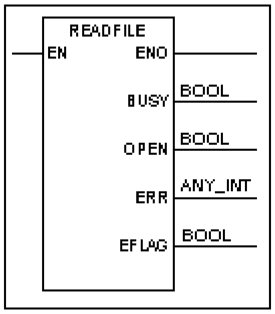
Other Trigger Function Blocks

ABORT_ALL	F_TRIG

READFILE

The READFILE function reads data from a file and stores it in a variable of a user-defined data type. The READFILE is one of eight function blocks that do file operations. Note that these function blocks are not designed for high-speed I/O execution or data transfers of large blocks of information.

Graphic representation:



Structured text syntax:

```
READFILE (FCB:=<file control block name>,
    OUT:=<variable>,[F:=<fieldsep>],[S:=<stringsep>],[T:=<e
    ol>]);
```

Parameter	Type	Description
File Control Block Name	FILE	Name of the file control block that handles operations for this file.
Variable	USER-DEFINED	Name of the user-defined data type variable to which the file data is stored. Any user-defined data type. If only one value needs to be read or written, you can use a variable of type ANY.
Fieldsep [1]	STRING	String character used to separate fields in the file. If you create the file using another application, such as a text editor instead of WRITEFILE, be sure to use field separators between values. Any valid STRING character. The default is the space character.
Stringsep [1]	STRING	String character used to delimit the strings in the file. The string delimiter is not required. However, if you create the file using another application, such as a text editor instead of the WRITEFILE, be sure that the data is formatted so that READFILE reads it correctly. Any valid STRING character. The default is the double-quotation mark character.

Parameter	Type	Description
EOL [1]	STRING	String character used to indicate the end of a line in the file. InControl treats the EOL delimiter as a field separator. This allows the READFILE to read from more than one line at a time. However, as a field separator, the EOL character prevents your starting a value on one line and continuing it on the next. Valid values: any valid STRING character. The default is the new line character.
BUSY [2]	BOOL	Indicates the file control block is busy. Any BOOL variable.
OPEN [2]	BOOL	Indicates the file has been opened. Any BOOL variable.
EFLAG [2]	BOOL	Indicates a file operation error has occurred. Any BOOL variable.
ERR [2]	ANY_INT	Contains the error code if a file operation error occurs. Any SINT, INT, or DINT, BYTE, WORD, DWORD variable.

1 Choose delimiters carefully to avoid conflicts with string characters contained within the file. See "WRITEFILE" for more information.
2 Entries in the output fields are optional. However, for each field there is a default file control variable. As you design your program, use these output variables to handle file control. These fields reflect outputs in the specified file control block and are not actual parameters to the READFILE function. For a detailed description of the file control variables, see "CLOSEFILE."

You also need to create a user-defined data type using a data structure appropriate for the data read from the file.

Operation is as follows.

- The OPENFILE or the NEWFILE must open the file before the READFILE can read it.
- READFILE reads the file that is associated with the file control block.
- READFILE stores the data to the user-defined variable.
- The file control variables handle access to the file and error conditions, as described for the "CLOSEFILE" function block. The three input file control variables are of particular importance when files are opened. You need to verify whether their default values are appropriate for your application. All file function blocks that operate on the same file must use the same File Control Block name.
- If an error occurs, the file error variable is set to TRUE and a message appears in the Output window and the Wonderware Logger. The graphical output ENO is set to FALSE. For a complete list of the error codes, see "CLOSEFILE."

Note that the WRITEFILE writes data of the TIME, DATE, DT, and TOD data types in STRING format instead of number format. The READFILE can read this data format, provided it is formatted correctly following the IEC-61131 specification.

For an example that uses the READFILE with several other File procedures, see "WRITEFILE."

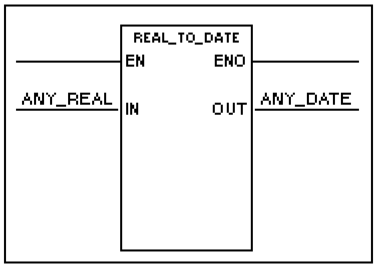
Other File Functions

CLOSEFILE	COPYFILE	DELETEFILE
NEWFILE	OPENFILE	REWINDFILE
WRITEFILE		

REAL_TO_DATE

The DATE_TO_REAL function converts an ANY_REAL input to a DATE value.

Graphical representation:



Structured text syntax:

```
<OUT> := REAL_TO_DATE (<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the value to convert. Any value, variable, or expression that resolves to an ANY_REAL data type (REAL, LREAL).
OUT	ANY_DATE	Contains the result of the conversion. An ANY_DATE data type (DT, DATE, TOD).

Operation is as follows:

- REAL_TO_DATE converts the value represented by the input variable and stores the result as an ANY_DATE data type in the output variable.

  The whole number portion of the real number represents the number of days since December 30, 1899. The fractional part of the number represents time of day.

- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Note To help ensure accuracy, use an LREAL data type for the input parameter when converting to a DT data type.

Examples of the REAL_TO_DATE operation:

REAL_TO_DATE(36885.25) returns DT#2000-12-25-06:00:00

REAL_TO_DATE(36885) returns DATE#2000-12-25

REAL_TO_DATE(0.75) returns TOD#18:00:00

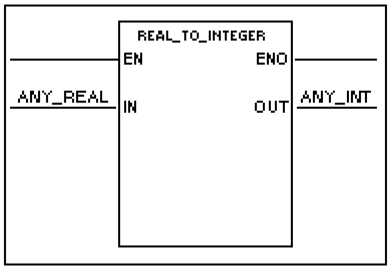
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	REAL_TO_DATE	REAL_TO_STRING
REAL_TO_TIME	STRING_TO_ARRAY	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

REAL_TO_INT

The REAL_TO_INT function converts an ANY_REAL input to an ANY_INT value.

Graphical representation:



Structured text syntax:

```
<OUT> := REAL_TO_INT (<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the data value to convert. An ANY_REAL data type (REAL, LREAL).
OUT	ANY_INT	Contains the result of the conversion. Any value, variable, or expression that resolves to an ANY_INT data type (SINT, INT, DINT, BYTE, WORD, DWORD).

Operation is as follows:

- REAL_TO_INT converts the value represented by the input variable and stores the result as an ANY_INT data type in the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the REAL_TO_INT operation:

REAL_TO_INT(4.99) returns 4

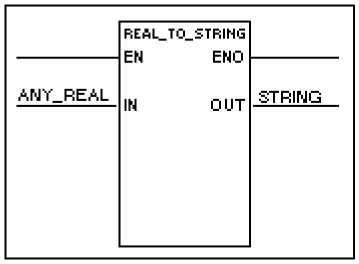
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	REAL_TO_DATE	REAL_TO_STRING
REAL_TO_TIME	STRING_TO_ARRAY	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

REAL_TO_STRING

The REAL_TO_STRING function converts an ANY_REAL input to a string.

Graphical representation:



Structured text syntax:

```
<OUT> := REAL_TO_STRING (<IN>);
```

Parameter	**Type**	**Description**
IN	ANY_REAL	Contains the data value to convert. An ANY_REAL data type (REAL, LREAL).
OUT	STRING	Contains the result of the conversion. A STRING variable data type.

Operation is as follows:

- REAL_TO_STRING converts the value represented by the input variable and stores the result as a STRING data type in the output variable.

  LREALs have up to 14 digits following the decimal point. REALs have up to six digits following the decimal point.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the REAL_TO_STRING operation:

REAL_TO_STRING(1.12345678901234) returns '1.12345678901234')

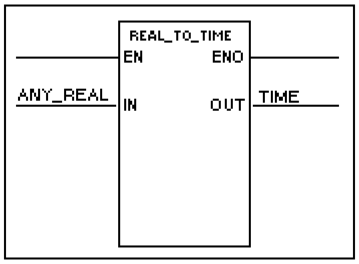
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	REAL_TO_DATE	REAL_TO_INT
REAL_TO_TIME	STRING_TO_ARRAY	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

REAL_TO_TIME

The REAL_TO_STRING function converts an ANY_REAL input to a TIME value.

Graphical representation:



Structured text syntax:

```
<OUT> := REAL_TO_TIME (<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the data value to convert. An ANY_REAL data type (REAL, LREAL).
OUT	TIME	Contains the result of the conversion. Any TIME variable data type.

Operation is as follows:

- REAL_TO_TIME converts the value represented by the input variable and stores the result as a TIME data type in the output variable.

  The real number represents the time seconds.

- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the REAL_TO_TIME operation:

REAL_TO_TIME(64.5) returns T#1m4s500ms

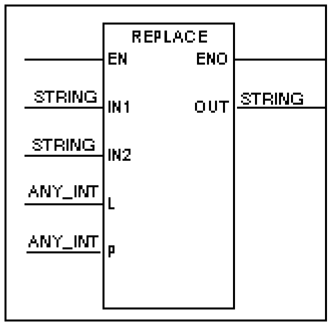
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	REAL_TO_DATE	REAL_TO_INT
REAL_TO_STRING	STRING_TO_ARRAY	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

REPLACE

The REPLACE function replaces the specified number of characters in a string with a set of characters from another string.

Graphical representation:



Structured text syntax:

`<OUT>:=REPLACE(IN1:=<IN1> ,IN2:=<IN2> ,L:= <L> ,P:= <P>);`

Parameter	Type	Description
IN1	STRING	Contains the string in which characters are replaced. Any valid STRING character or variable.
IN2	STRING	Contains string from which the characters are copied. Any valid STRING character or variable.
L	ANY_INT	Specifies the number of characters to replace. Any SINT, INT, DINT, BYTE, WORD, DWORD constant or variable.
P	ANY_INT	Specifies the position within the string to begin replacing characters. If P specifies a position outside the string, IN2 is concatenated to IN1. Any SINT, INT, DINT, BYTE, WORD, DWORD constant or variable.
OUT	STRING	Contains the result of the character replacement. Any valid STRING variable.

Operation is as follows:

- REPLACE replaces the number of characters specified by L in IN1 with characters from IN2.
- REPLACE replaces characters starting at position P.
- After the characters are replaced, REPLACE stores the new string in output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function. If an error occurs, e.g., L is not valid for the length of the string, zero is stored to OUT.

Example of the REPLACE operation:

REPLACE('BROWN', 'CD', 2, 3) returns 'BRCDN'

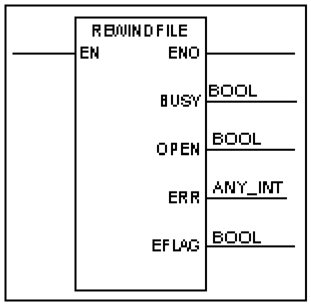
Other String Functions

CONCAT	DELETE	FIND
INSERT	LEFT	LEN
MID	MSGWND	RIGHT

REWINDFILE

The REWINDFILE function positions the internal file pointer to the beginning of a file. The REWINDFILE is one of eight procedures that do file operations. Note that these procedures are not designed for high-speed I/O execution, data transfers of large blocks of information, or for control applications.

Graphical representation:



Structured text syntax:

```
REWINDFILE (<fcb>);
```

Parameter	Type	Description
File Control Block Name	FILE	Name of file control block that handles operations for this file.
BUSY [1]	BOOL	Indicates the file control block is busy. Any BOOL variable.
OPEN [1]	BOOL	Indicates the file has been opened. Any BOOL variable.
EFLAG [1]	BOOL	Indicates a file operation error has occurred. Any BOOL variable.
ERR [1]	ANY_INT	Contains the error code if a file operation error occurs. Any SINT, INT, DINT, BYTE, WORD, DWORD variable.
1 Entries in the output fields are optional. However, for each field there is a default file control variable. As you design your program, you must use these output variables to handle file control. These fields reflect outputs in the specified file control block and are not actual parameters to the REWINDFILE function. For a detailed description of the file control variables, see "CLOSEFILE."		

Operation is as follows.

- REWINDFILE rewinds the file that is associated with the control block, specified in the File Control Block Name field.
- The file control variables handle access to the file and error conditions, as described for the CLOSEFILE function block. All file function blocks that operate on the same file must use the same File Control Block name.

- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.
- If an error occurs, the file error variable is set to TRUE and a message appears in the Output window and the Wonderware Logger. The graphical output ENO is set to FALSE. For a list of the error codes, see "CLOSEFILE."

You can do only one file operation for each File Control Block at a time. Note that file control I/O operations take place asynchronously to program execution.

For an example that uses the REWINDFILE with several other File procedures, see "WRITEFILE."

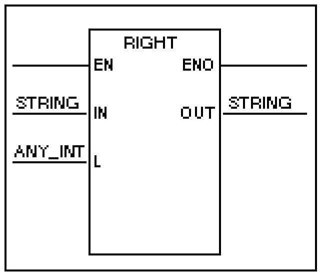
Other File Functions

CLOSEFILE	COPYFILE	DELETEFILE
NEWFILE	OPENFILE	READFILE
WRITEFILE		

RIGHT

The RIGHT function creates a string of characters from a specified number of the rightmost characters of another string of characters.

Graphical representation:



Structured text syntax:

```
<OUT> := RIGHT (IN:= <IN> , L:= <L>);
```

Parameter	Type	Description
IN	STRING	Contains the string from which the characters are copied. Any valid STRING character or variable.
L	ANY_INT	Specifies the number of characters to copy. Any SINT, INT, DINT, BYTE, WORD, DWORD constant or variable.
OUT	STRING	Contains the new string of characters. Any valid STRING variable.

Operation is as follows:

- RIGHT copies the number of characters specified by L to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function. Possible errors include a negative value in the Length field.

The following is an example of the RIGHT operation:

RIGHT('BROWN', 2) returns 'WN'

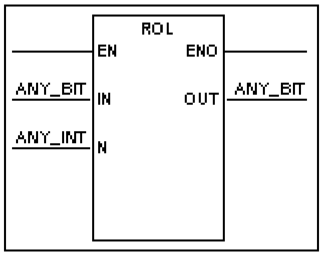
Other String Functions

CONCAT	DELETE	FIND
INSERT	LEFT	LEN
MID	MSGWND	REPLACE

ROL

The ROL function rotates the individual bits of an input a specified number of positions to the left.

Graphical representation:



Structured text syntax:

`<OUT> := ROL(IN:= <IN>, N:= <S>);`

Parameter	Type	Description
IN	ANY_BIT	Contains the value in which bits are rotated. Any BOOL, BYTE, WORD, DWORD constant or variable.
N	ANY_INT	Specifies the number of bit positions to rotate. Any SINT, INT, DINT constant or variable.
OUT	ANY_BIT	Contains the result of rotating the bits in IN. Any BOOL, BYTE, WORD, DWORD constant or variable.

Operation is as follows:

- ROL examines IN in its binary form and then shifts each bit to the left by the number of positions specified by N.
- The most significant bit shifts to the position of the least significant bit.
- ROL stores the result of the shift in the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function. Possible errors include a negative value in the N field.

You can also use this function with arrays as shown in the example below.

Example of the ROL operation:

When byte1 = 2#1111_0000

ROL(byte1) returns 2#1110_0001

Array1 := ROL(Array1,2);

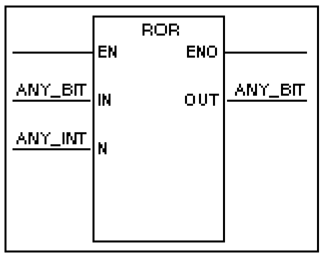
Other Bitwise Functions

AND	NOT	OR
ROR	SHL	SHR
XOR		

ROR

The ROR function rotates the individual bits of an input a specified number of positions to the right.

Graphical representation:



Structured text syntax:

```
<OUT> := ROR(IN:= <IN>, N:= <S>);
```

Parameter	Type	Description
IN	ANY_BIT	Contains the value in which bits are rotated. Any BOOL, BYTE, WORD, DWORD constant or variable.
N	ANY_INT	Specifies the number of bit positions to rotate. Any SINT, INT, DINT constant or variable.
OUT	ANY_BIT	Contains the result of rotating the bits in IN. Any BOOL, BYTE, WORD, DWORD constant or variable.

Operation is as follows:

- ROR examines input IN in its binary form and then shifts each bit to the right by the number of positions specified by N.
- The least significant bit shifts to the position of the most significant bit.
- ROR stores the result of the shift in the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function. Possible errors include a negative value in the N field.

You can also use this function with arrays as shown in the example below.

Example of the ROR operation:

When byte1 = 2##0000_1111

ROR(byte1) returns 2#1000_0111

Array1 := ROR(Array1,2);

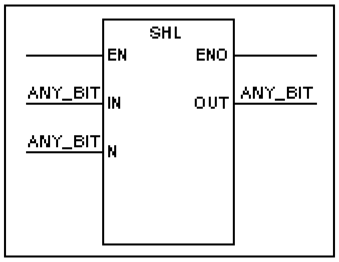
Other Bitwise Functions

AND	NOT	OR
ROL	SHL	SHR
XOR		

SHL

The SHL function shifts the individual bits of an input a specified number of positions to the left.

Graphical representation:



Structured text syntax:

```
<OUT> := SHL(<IN>, <N>);
```

Parameter	Type	Description
IN	ANY_BIT	Contains the value in which bits are shifted. Any BOOL, BYTE, WORD, DWORD constant or variable.
N	ANY_BIT	Specifies the number of bit positions to shift. Any SINT, INT, DINT constant or variable.
OUT	ANY_BIT	Contains the result of shifting bits in IN. Any BOOL, BYTE, WORD, DWORD variable.

Operation is as follows:

- SHL examines input IN in its binary form and then shifts each bit to the left by the number of positions specified by N.
- The most significant bit is lost after the shift.
- SHL stores the result of the shift in the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function..
- If N is negative or specifies a value that is greater than the number of bits in OUT, SHL sets OUT to zero.

You can also use this function with arrays as shown in the example below.

Example of the SHL operation.

When byte1 = 2#1111_0000

SHL(byte1) returns 2#1110_0000

Array1 := SHL(Array1,2);

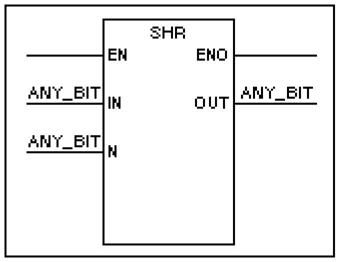
Other Bitwise Functions

AND	NOT	OR
ROL	ROR	SHR
XOR		

SHR

The SHR function shifts the individual bits of an input a specified number of positions to the right.

Graphical representation:



Structured text syntax:

```
<OUT> := SHR(<IN>, <N>);
```

Parameter	Type	Description
IN	ANY_BIT	Contains the value in which bits are shifted. Any BOOL, BYTE, WORD, DWORD constant or variable.
N	ANY_BIT	Specifies the number of bit positions to shift. Any SINT, INT, DINT constant or variable.
OUT	ANY_BIT	Contains the result of shifting bits in IN. Any BOOL, BYTE, WORD, DWORD variable.

Operation is as follows:

- SHR examines input IN in its binary form and then shifts each bit to the right by the number of positions specified by N.
- The least significant bit is lost after the shift.
- SHR stores the result of the shift in the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function..
- If N is negative or specifies a value that is greater than the number of bits in OUT, SHR sets OUT to zero.

You can also use this function with arrays as shown in the example below.

Example of the SHR operation.

When byte1 = 2#0000_1111

SHR(byte1) returns 2#0000_0111

Array1 := SHR(Array1,2);

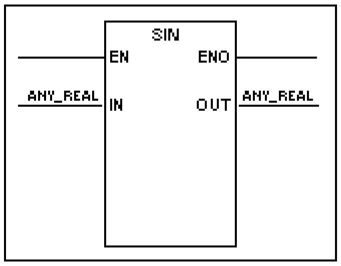
Other Bitwise Functions

AND	NOT	OR
ROL	ROR	SHL
XOR		

SIN

The SIN function calculates the sine of the input, which must be in radians.

Graphical representation:



Structured text syntax:

```
<OUT> := SIN(<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the value in radians for which the sine is calculated. Any REAL or LREAL constant or variable.
OUT	ANY_REAL	Contains the sine of IN. Any REAL or LREAL variable.

Operation is as follows:

- If the input IN (radians) is within the range of the selected data type, SIN stores the sine of IN to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Examples of the SIN operation:

SIN(0) returns 0

SIN(pi/2) returns 1

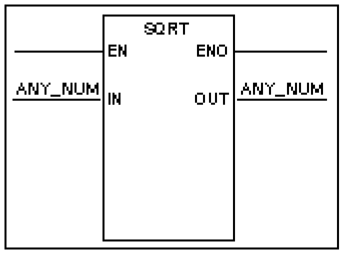
Other Trig/Log Functions

ACOS	ASIN	ATAN
COS	EXP	LN
LOG	TAN	

SQRT

The SQRT function calculates the square root of the input.

Graphical representation:



Structured text syntax:

```
<OUT> := SQRT(<IN>);
```

Parameter	Type	Description
IN	ANY_NUM	Contains the value for which the square root is calculated. IN must be positive. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
OUT	ANY_NUM	Contains the square root of the input. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD variable.

Operation is as follows:

- If the input IN is positive and within the range of the selected data type, SQRT calculates the square root of IN, stores the result to the output variable OUT
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function, such as when IN is negative

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

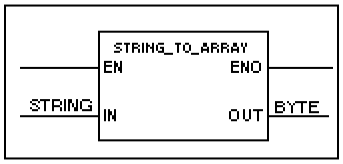
Other Math Functions

ABS	ADD	DIV
EXPT	MAX	MIN
MOD	MOVE	MUL
NEG	SUB	TRUNC

STRING_TO_ARRAY

The STRING_TO_ARRAY function stores an input consisting of a string of characters to an array of bytes.

Graphical representation:



Structured text syntax:

```
STRING_TO_ARRAY(OUT := <OUT> ,IN := <IN>);
```

Parameter	Type	Description
IN	STRING	Contains the string of characters to be converted. Any valid STRING character code or array of bytes containing valid STRING character codes. Values are decimal codes.
OUT	BYE	Contains the result of the conversion of the string to an array of bytes. Any BYTE variable array.

Operation is as follows:

- STRING_TO_ARRAY stores each ASCII character in the string IN to the output variable OUT a byte at a time.
- If the byte variable array is not large enough to hold every character in the string, plus an additional byte for the terminating 0, STRING_TO_ARRAY fills the array and stops. If the string is smaller than the array, the remaining elements of the array are set to zero.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the STRING_TO_ARRAY operation:

```
STRING_TO_ARRAY (OUT:= byte_array, IN:= string1);
```

If string1 contains ’Sara’, then the array consists of five bytes, and byte_array[0] = 83, byte_array[1] = 97, byte_array[2] = 114, byte_array[3] = 97, and byte_array[4] = 0.

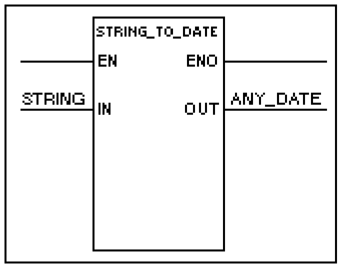
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	REAL_TO_DATE	REAL_TO_INT
REAL_TO_STRING	REAL_TO_TIME	STRING_TO_DATE
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

STRING_TO_DATE

The STRING_TO_DATE function converts an input string to a DATE value.

Graphical representation:



Structured text syntax:

```
<OUT> := STRING_TO_DATE (<IN>);
```

Parameter	Type	Description
IN	STRING	Contains the string to convert. A variable (STRING data type). The contents of IN must match the format of the DATE, DT, or TOD data types.
OUT	ANY_DATE	Contains the result of the conversion. Any value, variable, or expression that resolves to an ANY_DATE data type (DT, DATE, TOD).

Operation is as follows:

- STRING_TO_DATE converts the value represented by the input variable and stores the result as an ANY_DATE data type in the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.
- If the contents of IN do not match the format of the DATE data type, OUT is set to zero.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the STRING_TO_DATE operation:

STRING_TO_DATE('DT#2000-12-25-06:00:00')
returns DT#2000-12-25-06:00:00

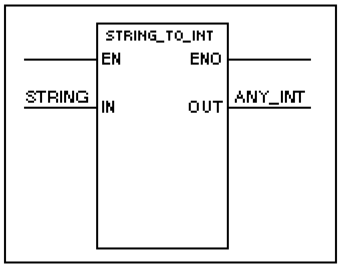
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	REAL_TO_DATE	REAL_TO_INT
REAL_TO_STRING	REAL_TO_TIME	STRING_TO_ARRAY
STRING_TO_INT	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

STRING_TO_INT

The STRING_TO_INT function converts an input string to an ANY_INT value.

Graphical representation:



Structured text syntax:

```
<OUT> := STRING_TO_INT (<IN>);
```

Parameter	Type	Description
IN	STRING	Contains the string to convert. Any STRING, or a variable or expression that resolves to a STRING data type.
OUT	ANY_INT	Contains the result of the conversion. An ANY_INT data type (SINT, INT, DINT, BYTE, WORD, DWORD).

Operation is as follows:

- STRING_TO_INT converts the value represented by the input variable and stores the result as an ANY_INT data type in the output variable.
- If the converted value cannot be represented as a DINT or DWORD data type, OUT is set to zero.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Examples of the STRING_TO_INT operation:

STRING_TO_INT('123') returns 123

STRING_TO_INT('12ABC') returns 12

STRING_TO_INT('ABC5') returns 0

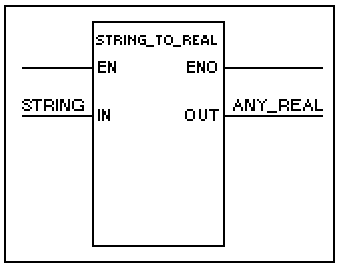
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	REAL_TO_DATE	REAL_TO_INT
REAL_TO_STRING	REAL_TO_TIME	STRING_TO_ARRAY
STRING_TO_DATE	STRING_TO_REAL	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

STRING_TO_REAL

The STRING_TO_REAL function converts a string input to an ANY_REAL value.

Graphical representation:



Structured text syntax:

```
<OUT> := STRING_TO_REAL (<IN>);
```

Parameter	Type	Description
IN	STRING	Contains the data value to convert. Any STRING, or a variable or expression that resolves to a STRING data type.
OUT	ANY_REAL	Contains the result of the conversion. An ANY_REAL data type (REAL, LREAL).

Operation is as follows:

- STRING_TO_REAL converts each numeric character, up to the first non-real character, in the string input IN to its decimal value and stores the result of the conversion in OUT. If the string does not represent a valid REAL value, STRING_TO_REAL stores zero in the output variable.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the STRING_TO_REAL operation:

STRING_TO_REAL('3.141') returns 3.141

STRING_TO_REAL('2.71ABC') returns 2.71

STRING_TO_REAL('ABC2) returns 0

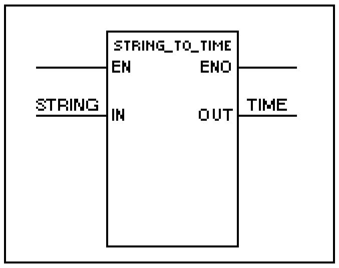
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	REAL_TO_DATE	REAL_TO_INT
REAL_TO_STRING	REAL_TO_TIME	STRING_TO_ARRAY
STRING_TO_DATE	STRING_TO_INT	STRING_TO_TIME
TIME_TO_REAL	TIME_TO_STRING	

STRING_TO_TIME

The STRING_TO_TIME function converts a string input to a TIME value.

Graphical representation:



Structured text syntax:

```
<OUT> := STRING_TO_TIME (<IN>);
```

Parameter	Type	Description
IN	STRING	Contains the data value to convert. A STRING or a variable or expression that resolves to a STRING data type. The contents of IN must match the format of the TIME data type.
OUT	TIME	Contains the result of the conversion. A TIME data type.

Operation is as follows:

- STRING_TO_TIME converts each character in the string IN to its decimal value and stores the result in the output variable OUT according to the TIME data type, described above.
- If the contents of IN do not match the format of the TIME data type, OUT is set to zero.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the STRING_TO_TIME operation:

STRING_TO_TIME('T#33s') returns T#33

STRING_TO_TIME('64.5') returns T#1m4s500ms

STRING_TO_TIME('T#4.2m') returns T#4m12s

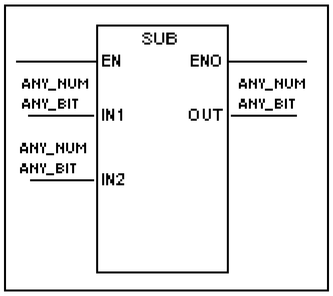
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	REAL_TO_DATE	REAL_TO_INT
REAL_TO_STRING	REAL_TO_TIME	STRING_TO_ARRAY
STRING_TO_DATE	STRING_TO_INT	STRING_TO_REAL
TIME_TO_REAL	TIME_TO_STRING	

SUB

The SUB function subtracts one input value from another.

Graphical representation:



Structured text syntax:

```
<OUT> := <IN1> -<IN2>;
```

Parameter	Type	Description
IN1	ANY_NUM ANY_BIT [1]	Contains the minuend, the number from which a value is subtracted. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
IN2	ANY_NUM ANY_BIT [1]	Contains the subtrahend, the number that is subtracted. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD constant or variable.
OUT	ANY_NUM ANY_BIT [1]	Contains the result of the subtraction of IN2 from IN1. Any SINT, INT, DINT, REAL, LREAL, BYTE, WORD, DWORD variable.
1 BOOL data types are not allowed.		

Operation is as follows:

- SUB subtracts IN2 from IN1 and stores the result to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

You can also use this function with arrays. For example, you can subtract a number from every element in an array with the following line:

Array1 := Array2 - 9;

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

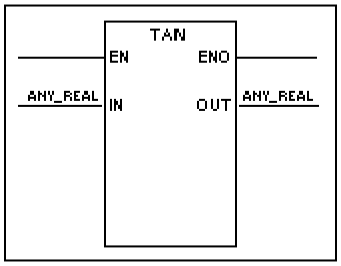
Other Math Functions

ABS	ADD	DIV
EXPT	MAX	MIN
MOD	MOVE	MUL
NEG	SQRT	TRUNC

TAN

The TAN function calculates the tangent of the input, which must be in radians.

Graphical representation:



Structured text syntax:

```
<OUT> := TAN(<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the value in radians for which the tangent is calculated. Any REAL or LREAL constant or variable.
OUT	ANY_REAL	Contains the tangent of IN. Any REAL or LREAL variable.

Operation is as follows:

- If the input IN (radians) is within the range of the selected data type, TAN stores the tangent of IN to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Examples of the TAN operation:

TAN(pi/4) returns 1

TAN(0) returns 0

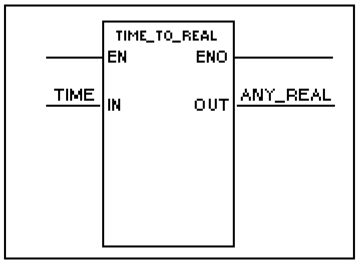
Other Trig/Log Functions

ACOS	ASIN	ATAN
COS	EXP	LN
LOG	SIN	

TIME_TO_REAL

The TIME_TO_REAL function converts a TIME input to an ANY_REAL value.

Graphical representation:



Structured text syntax:

```
<OUT> := TIME_TO_REAL (<IN>);
```

Parameter	Type	Description
IN	TIME	Contains the data value to convert. A value, variable, or expression that resolves to a TIME data type.
OUT	ANY_REAL	Contains the result of the conversion. An ANY_REAL data type (REAL, LREAL).

Operation is as follows:

- TIME_TO_REAL converts the value represented by the input variable and stores the result as an ANY_REAL data type in the output variable.

  The real number represents the time seconds.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the TIME_TO_REAL operation:

TIME_TO_REAL(T#1m4s500ms) returns 64.5

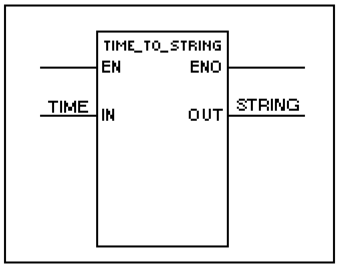
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	REAL_TO_DATE	REAL_TO_INT
REAL_TO_STRING	REAL_TO_TIME	STRING_TO_ARRAY
STRING_TO_DATE	STRING_TO_INT	STRING_TO_REAL
STRING_TO_TIME	TIME_TO_STRING	

TIME_TO_STRING

The TIME_TO_STRING function converts a TIME input to a string.

Graphical representation:



Structured text syntax:

```
<OUT> := TIME_TO_STRING (<IN>);
```

Parameter	Type	Description
IN	TIME	Contains the data value to convert. A value, variable, or expression that resolves to a TIME data type.
OUT	STRING	Contains the result of the conversion. Any STRING variable.

Operation is as follows:

- TIME_TO_STRING converts the content of IN to ASCII characters and stores them in the string OUT using the format of the TIME data type.
- If the contents of IN do not match the format of the TIME data type, OUT is set to zero.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

Example of the TIME_TO_STRING operation:

TIME_TO_STRING(T#1m4s500ms) returns the string 'T#1m4s500ms'

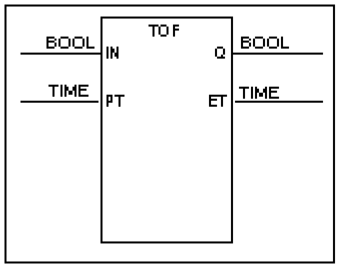
Other Conversion Functions

ARRAY_TO_STRING	BCD_TO_INT	DATE_TO_REAL
DATE_TO_STRING	INT_TO_BCD	INT_TO_REAL
INT_TO_STRING	REAL_TO_DATE	REAL_TO_INT
REAL_TO_STRING	REAL_TO_TIME	STRING_TO_ARRAY
STRING_TO_DATE	STRING_TO_INT	STRING_TO_REAL
STRING_TO_TIME	TIME_TO_REAL	

TOF

The TOF function block times the duration of an event. After timing up to the preset interval, the TOF turns off an output, which makes the TOF an off-delay timer.

Graphical representation:



Structured text syntax:

```
<TOF Name>(IN:=<timer input>, PT:=<preset time>,
   EN:=<enable>, Q:=<output>,ET:=<elapsed time>,
   ENO:=<enable output>);
```

Parameter	Type	Description
TOF Name	TOF	Unique name for the timer.
IN	BOOL	Input starts the timer. Any BOOL variable.
PT	TIME	Specifies the period for which the timer times. Any TIME data type.
EN	BOOL	Enables the timer. Any BOOL variable.
Q	BOOL	Output changes to FALSE when timer times out. Any BOOL variable.
ET	TIME	Contains the current elapsed time. Any TIME data type.
ENO	BOOL	Echoes EN. Any BOOL variable.

To set PT, enter the duration directly or click on **Define Time Duration** and enter time intervals in the dialog box.

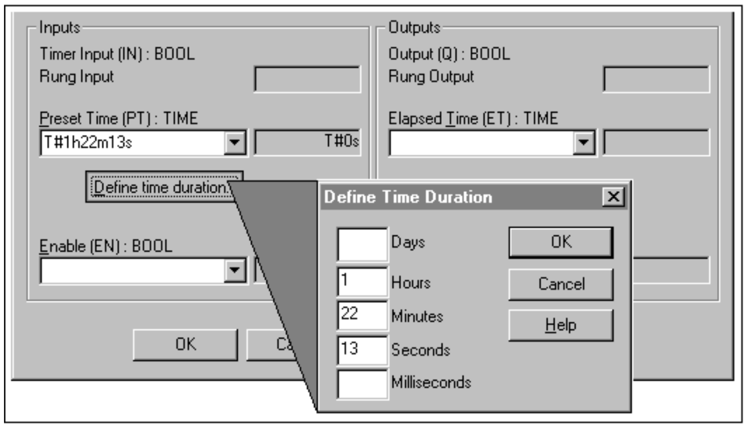
- If you enter the duration directly, follow the IEC 61131-3 specification: a keyword, e.g., T#, TIME#, t#, time#, followed by time in days, hours, minutes, seconds. Examples are shown below.

Timer Duration Examples

Time	Format	Time	Format
14.7 days	T#14.7d	4 seconds	Time#4s
2 minutes and 5 seconds	T#2m5s	1 day 29 minutes	t#1d29m
74 minutes*	time#74m	1 hour 5 seconds and 44 milliseconds	T#1h5s44ms
* The IEC 61131-3 specification allows overflow of the most significant unit in a duration.			

- If you prefer to use the dialog box, enter the time into each field as appropriate.

The figure shown below illustrates the same time entered by both methods.



Setting Timer Duration

Operation is as follows.

- When the enable input EN is TRUE the timer is enabled.

  When EN is FALSE, the timer is not enabled and cannot begin timing.

  When EN is set to FALSE, all timer outputs are set to FALSE.
- When input IN transitions from TRUE to FALSE, the timer begins timing, storing the elapsed time in output ET.
- While the timer is timing, rung output Q is TRUE.
- If EN is set to FALSE while the timer is timing, ET is frozen in its current state; Q is set to FALSE.

  If EN changes from FALSE back to TRUE, the timer resumes timing, beginning at the current value stored in ET; Q is set back to TRUE.
- When the elapsed time ET equals preset time PT, the timer sets Q to FALSE.
- If input IN is set to TRUE, the timer resets ET to zero and stops timing; Q remains TRUE.
- Enable output ENO echoes the value of EN.

WARNING! Assigning the same function block name to different timers may cause unpredictable operation by the controller, which can result in death or injury to personnel and/or damage to equipment. Always use a unique name for each timer.

When the program is running, you can double-click a TOF to display the TOF dialog box. The box displays the current value of all function block inputs and outputs. You can also open the Watch window and enter timer variables that you can observe at runtime.

Note Timers evaluate actual time elapsed and are not affected by setting a program or the runtime engine to Paused mode.

You can use any of the TOF inputs and outputs in any expression, contact or coil instead of a symbol of the same type. To reference a TOF input or output, enter the function block name followed by a period and the specific input or output suffix. For example, TOF5.Q refers to the rung output of TOF5. TOF variables are listed below.

Variable	Reference Name
IN	xxx.IN
PT	xxx.PT
EN	xxx.EN
Q	xxx.Q
ET	xxx.ET
ENO	xxx.ENO
Note xxx denotes the timer function block name.	

An example of the TOF timing diagram is shown below:

A. EN and ENO have previously been set to TRUE. IN transitions from TRUE to FALSE and ET indicates the timer has begun timing.

B. ET equals PT and Q transitions from TRUE to FALSE.

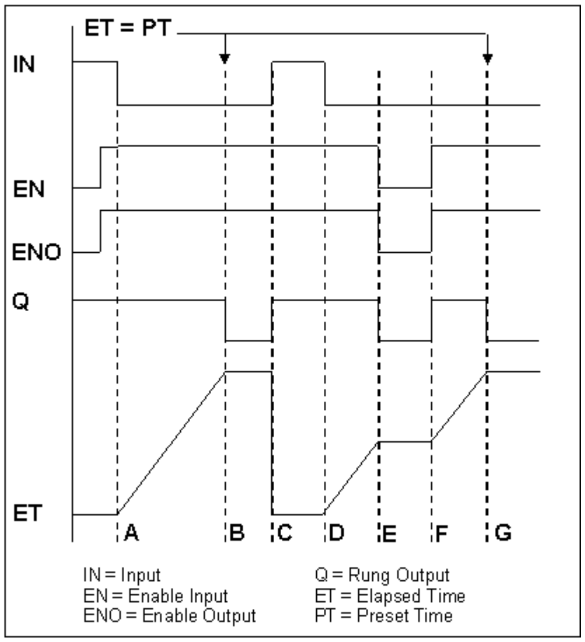
C. IN transitions from FALSE to TRUE.
Q is set to TRUE and ET is set to zero.

D. IN transitions from TRUE to FALSE and ET indicates the timer has begun timing.

E. EN transitions from TRUE to FALSE, disabling the timer.
ET, which had been timing, holds at its current value, and Q is set to FALSE.

F. EN transitions from FALSE to TRUE, re-enabling the timer.
ET resumes timing, and Q is set to TRUE.

G. ET equals PT and Q transitions from TRUE to FALSE.



TOF Timing Diagram

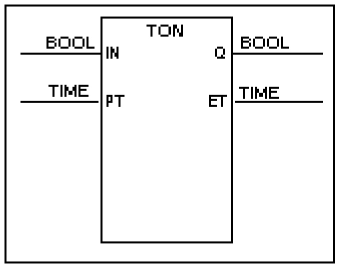
Other Timer Function Blocks

TON	TP

TON

The TON function block times the duration of an event. After timing up to the preset interval, the TON turns on an output, which makes the TON an on-delay timer.

Graphical representation:



Structured text syntax:

```
<TON Name>(IN:= <timer input>, PT:=<preset time>,
   EN:=<enable>, Q:=<output>,ET:=<elapsed
   time>,ENO:=<enable output>);
```

Parameter	Type	Description
TON Name	TON	Unique name for the timer.
IN	BOOL	Input starts the timer. Any BOOL variable.
PT	TIME	Specifies the period for which the timer times. Any TIME data type.
EN	BOOL	Enables the timer. Any BOOL variable.
Q	BOOL	Rung output changes to TRUE when timer times out. Any BOOL variable.
ET	TIME	Contains the current elapsed time. Any TIME data type.
ENO	BOOL	Echoes EN. Any BOOL variable.

WARNING! Assigning the same function block name to different timers may cause unpredictable operation by the controller, which can result in death or injury to personnel and/or damage to equipment. Always use a unique name for each timer.

To set PT, enter the duration directly or click on **Define Time Duration** and enter time intervals in the dialog box.

- If you enter the duration directly, follow the IEC 61131-3 specification: a keyword, e.g., T#, TIME#, t#, time#, followed by time in days, hours, minutes, seconds.

- If you prefer to use the dialog box, enter the time into each field as appropriate.

  For examples of setting duration, see "TOF."

Operation is as follows.

- When the enable input EN is TRUE, the timer is enabled.

  When EN is FALSE, the timer is not enabled and cannot begin timing.

  When EN is set to FALSE, all timer outputs are set to FALSE.
- When input IN transitions from FALSE to TRUE, the timer begins timing, storing the elapsed time in output ET.
- While the timer is timing, output Q is FALSE.
- If EN is set to FALSE while the timer is timing, ET is frozen in its current state; Q remains FALSE.

  If EN changes from FALSE back to TRUE, the timer resumes timing, beginning at the current value stored in ET; Q remains FALSE.
- When the elapsed time ET equals preset time PT, the timer sets Q to TRUE.
- If input IN is set to FALSE before ET equals PT, the timer resets ET to zero and stops timing; Q remains FALSE.
- Enable output ENO echoes the value of EN.

When the program is running, you can double click a TON to display the TON dialog box. The box displays the current value of all function block inputs and outputs. You can also open the Watch window and enter timer variables that you can observe at runtime.

Note Timers evaluate actual time elapsed and are not affected by setting a program or the runtime engine to Paused mode.

You can use any of the TON inputs and outputs in any expression, contact or coil instead of a symbol of the same type. To reference a TON input or output, enter the function block name followed by a period and the specific input or output suffix. For example, TON1.ET refers to the elapsed time output of TON1. All TON variables are listed below.

Variable	**Reference Name**
IN	xxx.IN
PT	xxx.PT
EN	xxx.EN
Q	xxx.Q
ET	xxx.ET
ENO	xxx.ENO
Note xxx denotes the timer function block name.	

An example of the TON timing diagram is shown below:

A. EN and ENO have been previously been set to TRUE.
IN transitions from FALSE to TRUE and ET indicates the timer has begun timing.

B. ET equals PT and Q transitions from FALSE to TRUE.

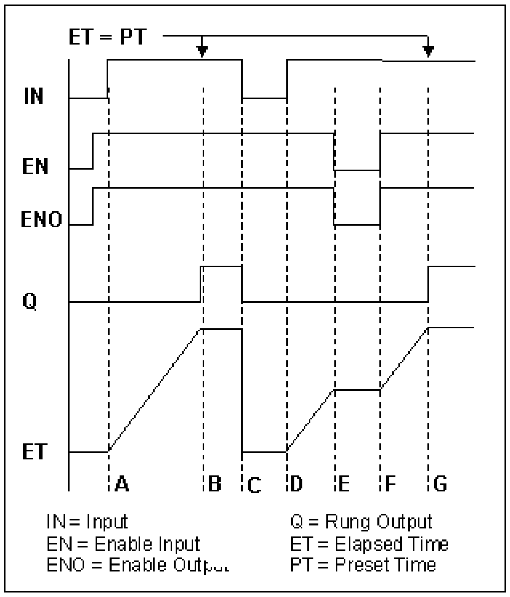
C. IN transitions from TRUE to FALSE.
Q is set to FALSE and ET is set to zero.

D. IN transitions from FALSE to TRUE and ET indicates the timer has begun timing.

E. EN transitions from TRUE to FALSE, disabling the timer.
ET, which had been timing, holds at its current value.

F. EN transitions from FALSE to TRUE, re-enabling the timer.
ET resumes timing.

G. ET equals PT and Q transitions from FALSE to TRUE.



TON Timing Diagram

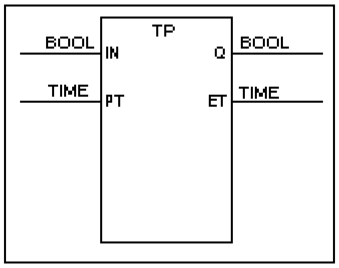
Other Timer Function Blocks

TOF	TP

TP

The TP function block times the duration of an event. After its input pulses from off to on, the TP times up to the preset interval and turns off an output, which makes the TP an off-delay timer.

Graphical representation:



Structured text syntax:

```
<TP Name>(IN:=<timer input>, PT:=<preset time>,
    EN:=<enable>, Q:=<output>, ET:=<elapsed time>,
    ENO:=<enable output>);
```

Parameter	Type	Description
TP Name	TP	Unique name for the timer.
IN	BOOL	Input starts the timer. Any BOOL variable.
PT	TIME	Specifies the period for which the timer times. Any TIME data type.
EN	BOOL	Enables the timer. Any BOOL variable.
Q	BOOL	Output changes to FALSE when timer times out. Any BOOL variable.
ET	TIME	Contains the current elapsed time. Any TIME data type.
ENO	BOOL	Echoes EN. Any BOOL variable.

WARNING! Assigning the same function block name to different timers may cause unpredictable operation by the controller, which can result in death or injury to personnel and/or damage to equipment. Always use a unique name for each timer.

To set PT, you can enter the duration directly or click on **Define Time Duration** and enter time intervals in the dialog box.

- If you enter the duration directly, follow the IEC 61131-3 specification: a keyword, e.g., T#, TIME#, t#, time#, followed by time in days, hours, minutes, seconds.

- If you prefer to use the dialog box, enter the time into each field as appropriate.

  For examples of setting duration, see "TOF."

Operation is as follows.

- When the enable input EN is TRUE, the timer is enabled.

  When EN is FALSE, the timer is not enabled and cannot begin timing.

  When EN is set to FALSE, all timer outputs are set to FALSE.
- When input IN transitions from FALSE to TRUE, the timer begins timing, storing the elapsed time in output ET.
- While the timer is timing, output Q is TRUE.
- If EN is set to FALSE while the timer is timing, ET is frozen in its current state; Q is set to FALSE.

  If EN changes from FALSE back to TRUE, the timer resumes timing, beginning at the current value stored in ET; Q is set back to TRUE.
- When the elapsed time ET equals preset time PT, the timer sets Q to FALSE.
- If input IN is set to FALSE before ET equals PT, the timer continues timing until ET equals PT.
- Enable output ENO echoes the value of EN.

When the program is running, you can double-click a TP to display the TP dialog box. The box displays the current value of all function block inputs and outputs. You can also open the Watch window and enter timer variables that you want to observe at runtime.

Note Timers evaluate actual time elapsed and are not affected by setting a program or the runtime engine to Paused mode.

You can use any of the TP inputs and outputs in any expression, contact or coil instead of a symbol of the same type. To reference a TP input or output, enter the function block name followed by a period and the specific input or output suffix. For example, TP9.IN refers to the input of TP9. All TP variables are listed in the following table.

Variable	**Reference Name**
IN	xxx.IN
PT	xxx.PT
EN	xxx.EN
Q	xxx.Q
ET	xxx.ET
ENO	xxx.ENO
Note xxx denotes the timer function block name.	

An example of the TP timing diagram is shown below:

A. EN and ENO have been previously been set to TRUE.
IN transitions from FALSE to TRUE, Q is set to TRUE, and ET indicates the timer has begun timing.
While the timer is timing, IN can toggle without affecting ET or Q.

B. ET equals PT and Q transitions from TRUE to FALSE.

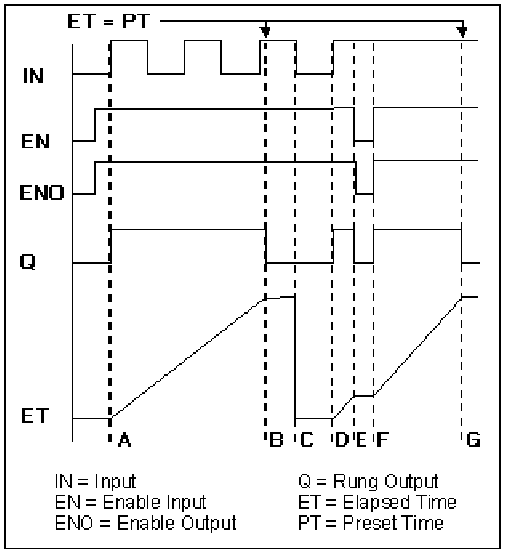
C. IN transitions from TRUE to FALSE.
ET is set to zero

D. IN transitions from FALSE to TRUE, Q is set to TRUE, and ET indicates that the timer has begun timing.

E. EN transitions from TRUE to FALSE, disabling the timer.
ET, which had been timing, holds at its current value, and Q is set to FALSE.

F. EN transitions from FALSE to TRUE, re-enabling the timer.
ET resumes timing., and Q is set to TRUE.

G. ET equals PT and Q transitions from TRUE to FALSE.



TP Timing Diagram

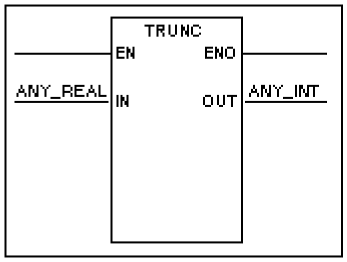
Other Timer Function Blocks

TOF TON

TRUNC

The TRUNC function removes one or more of the least significant digits of an ANY_REAL data type.

Graphical representation:



Structured text syntax:

```
<OUT> := TRUNC(<IN>);
```

Parameter	Type	Description
IN	ANY_REAL	Contains the value to be truncated. Any number, variable, or expression that resolves to an ANY_REAL data type (REAL, LREAL).
OUT	ANY_INT	Contains the truncated value of IN. An ANY_INT variable data type (SINT, INT, DINT, BYTE, WORD, DWORD).

Operation is as follows:

- TRUNC removes the fractional part of **<IN>** and stores the resulting integer value in **<OUT>**.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

For more information about using data types in math expressions, see "Variable Data Types" in the "Defining Variables" chapter of the *InControl Environment Manual*.

Example of the TRUNC operation:

TRUNC(4.9) returns 4.

TRUE AND FALSE returns FALSE.

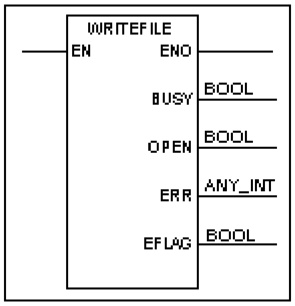
Other Math Functions

ABS	ADD	DIV
EXPT	MAX	MIN
MOD	MOVE	MUL
NEG	SQRT	SUB

WRITEFILE

The WRITEFILE function block writes data from a variable of a user-defined data type to a file. The WRITEFILE is one of eight function blocks that do file operations. Note that these function blocks are not designed for high-speed I/O execution or data transfers of large blocks of information.

Graphical representation:



Structured text syntax:

```
WRITEFILE (FCB:= <file control block name>, IN:=
    <variable>, [F:=<fieldsep>],
    [S:=<stringsep>],[T:=<eol>]);;
```

Parameter	Type	Description
File Control Block Name	FILE	Name of the file control block that handles operations for this file.
Variable	USER-DEFINED	Name of the user-defined data type variable containing the data that is written to the file. Any user-defined data type. If only one value needs to be read or written, you can use a variable of type ANY.
Field Separator [1]	STRING	String character used to separate fields in the file. During file write operations, InControl places a field separator between values. Any valid STRING character. The default is the space character.
String Delimiter [1]	STRING	String character used to separate strings in the file. FWRIT places the string delimiter on each end of a string prior to writing to the file. Any valid STRING character. The default is the double-quotation mark character.
EOL Delimiter [1]	STRING	String character used to indicate the end of a line in the file. A line can be up to 1 K in length. You cannot write more than 1 K of data for each execution of the FWRIT. When a write operation is finished and all values are written to the file, InControl attaches an EOL delimiter Any valid STRING character. The default is the new line character.

Parameter	Type	Description
BUSY [2]	BOOL	Indicates the file control block is busy. Any BOOL variable.
OPEN [2]	BOOL	Indicates the file has been opened. Any BOOL variable.
EFLAG [2]	BOOL	Indicates a file operation error has occurred. Any BOOL variable.
ERR [2]	ANY_INT	Contains the error code if a file operation error occurs. Any SINT, INT, DINT, BYTE, WORD, DWORD variable.

1 When outputting symbols of data type STRING, FWRIT writes IEC special characters ($N, $P, etc.) in their actual ASCII format. For example, $N generates a carriage return / linefeed in the resultant file. If you intend to reread the file using FREAD, choose delimiters carefully to avoid conflicts with string characters contained within the file.

2 Entries in the output fields are optional. However, for each field there is a default file control variable. As you design your program, you must use these output variables to handle file control. These fields reflect outputs in the specified file control block and are not actual parameters to the WRITEFILE function. For a detailed description of the file control variables, see "CLOSEFILE."

Operation is as follows.

- The OPENFILE or the NEWFILE must open the file before the WRITEFILE can write to it.
- WRITEFILE writes to the file that is associated with the control block, specified in the File Control Block Name field.
- The file control variables handle access to the file and error conditions, as described for the "CLOSEFILE" function block. All file function blocks that operate on the same file must use the same File Control Block name.
- The system writes the data from the user-defined data type specified in the Data field to the file.
- If an error occurs, the file error variable is set to TRUE, ENO is set to FALSE, and a message appears in the Output window and the Wonderware Logger. For a list of the error codes, see "CLOSEFILE."

You also need to create a user-defined data type variable using a data structure appropriate for the data read from the file.

Note that data of the TIME, DATE, DT, and TOD data types is written out in STRING format instead of number format. The READFILE can read this data format, provided it is formatted correctly following the IEC-61131 specification.

If you use WRITEFILE to append data to a file (the file was opened with the File Control Append Input Variable set to TRUE), then subsequent read operations on the file will generate an End of File error (code 31). Be sure to rewind the file before attempting to read it again.

The following is an example of the WRITEFILE procedure. Note also the usage of the NEWFILE, REWINDFILE, READFILE, and CLOSEFILE procedures.

```
NEWFILE (rpt, "my_file");
WRITEFILE (rpt, data_in);
REWINDFILE (rpt);
READFILE (FCB:= rpt, OUT:= data_out);
CLOSEFILE (rpt);
```

1. NEWFILE creates a file called my_file. The default location for my_file is the same directory where RTEngine.exe is located.
2. WRITEFILE copies the contents of the STRING variable data_in to my_file.
3. REWINDFILE sets the file pointer to the beginning of my_file.
4. READFILE copies the contents of my_file to the STRING variable data_out.
5. CLOSEFILE closes my_file.

Caution! Each file write operation prints one or more lines of ASCII text to a file. Writing data to the file multiple times can potentially corrupt the file contents, with the risk of losing information. This can occur when you rewind the file and in the next write operation, the length of a new line of information is different from the line that it overwrites. When you design your file write operations, make sure that new lines of data are the same length as the lines being replaced, or delete a file and replace it.

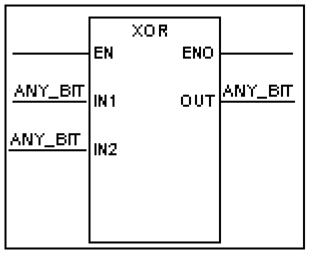
Other File Function Blocks

CLOSEFILE	COPYFILE	DELETEFILE
NEWFILE	OPENFILE	READFILE
REWINDFILE		

XOR

The XOR function block does a bitwise logical Exclusive OR of two values.

Graphical representation:



Structured text syntax:

```
<OUT> := <IN1> XOR <IN2>;
```

Parameter	Type	Description
IN1	ANY_BIT	Contains the first value to be XORed. Any BOOL, BYTE, WORD, DWORD constant or variable.
IN2	ANY_BIT	Contains the second value to be XORed. Any BOOL, BYTE, WORD, DWORD constant or variable.
OUT	ANY_BIT	Contains the result of XORing IN1 and IN2. Any BOOL, BYTE, WORD, DWORD variable.

Operation is as follows:

- XOR does an Exclusive OR on each bit of the two inputs IN1and IN2.
- XOR stores the result of the XOR operation to the output variable OUT.
- The EN BOOL parameter is used only in the graphical languages to enable the function to execute. ENO follows EN unless an error condition occurs within the function.

You can also use this function with arrays as shown in the example below.

Examples of the XOR operation:

2#0011 XOR 2#0101 returns 2#0110

TRUE XOR TRUE returns FALSE

Array1 := Array2 XOR #16FFFE;

Other Bitwise Functions

AND	NOT	OR
ROL	ROR	SHL
SHR		

Index

Wonderware®

InControl™ and InTouch® Reference User's Guide

Revision H

Last Revision: January 2004

Invensys Systems, Inc.

Invensys Systems, Inc.
26561 Rancho Parkway South
Lake Forest, CA 92630 U.S.A.
(949) 727-3200
http://www.wonderware.com

Contents

C H A P T E R 1

InControl and InTouch

This chapter describes the functions of the InControl wizards used in the InTouch environment.

Contents

InControl Functions Supported by InTouch

InControl provides the following six wizards that can be placed on an InTouch window. These wizards allow easy and effective interaction between InControl and InTouch.

- InControl Project Enables the operator to launch an InControl project. This starts InControl for the specified project and allows the operator to use all the InControl functions to edit, compile, download, and run the programs in that project. The InControl development environment must be installed on the operator’s hardware system.
- Configure Runtime Engine Enables the operator to make online configuration changes to the runtime engine on the specified node.
- InControl Mode Enables the operator to set the mode (Run, Pause, Single Scan) for the project that is downloaded to the specified node.
- InControl Edit Enables the operator to launch an individual program in a project. This starts InControl within the development environment, at a specified line within the specified program. The operator can use all the tools available in the development environment to edit, compile, download, and run the program. The InControl development environment must be installed on the operator’s hardware system.
- InControl Clear Faults Enables the operator to clear any runtime engine faults on the specified node at runtime.
- InControl Runtime Add Tag Links InTouch tags with InControl symbols (variables). The InControl development environment must be installed on the operator’s hardware system.

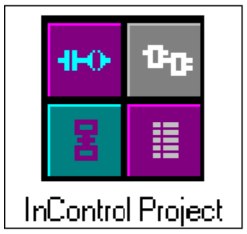
Installing the InControl Wizards

For instructions about adding the InControl wizards to the InTouch toolbox and placing them on an InTouch window, see the *InTouch User’s Guide*.

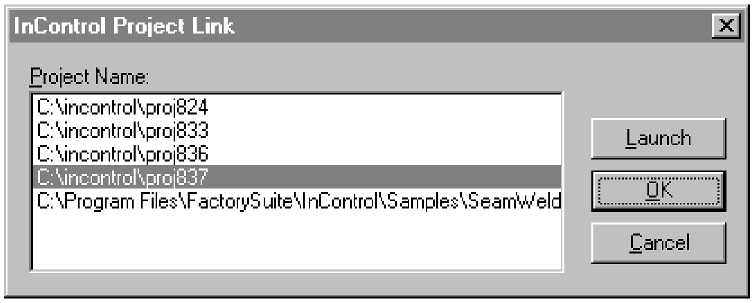
To install InControl wizards, see "Installation Guidelines" of the "Getting Started with InControl" chapter.

Using the InControl Project Wizard

Place the InControl Project wizard on an InTouch window to enable the operator to launch a project at runtime. Clicking this wizard from WindowViewer starts InControl within the runtime environment for the specified project. The operator can use all the functions of the InControl development environment to edit, validate, download, and run the programs in that project. The InControl development environment must be installed on the operator's hardware system.



If you double-click the InControl Project wizard from the InTouch WindowMaker, the **InControl Project Link** dialog box appears.



Project Name:

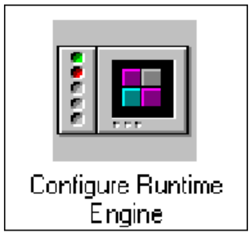
C:\incontrol\proj824
C:\incontrol\proj833
C:\incontrol\proj836
C:\incontrol\proj837
C:\Program Files\FactorySuite\InControl\Samples\SeamWeld

InControl Project Link Dialog Box

Button / Field	Description
Project Name	Select the default project to be launched in the Development environment. The operator can choose to launch any of the other projects that appear in the **Project Name** field. The InControl development environment must be installed on the operator's hardware system before the operator can edit any project files.
Launch	Click **Launch** to run the specified InControl project from WindowMaker.
OK	Accepts the selected project to be launched in the Development environment.
Cancel	Closes the wizard dialog box without saving entries.

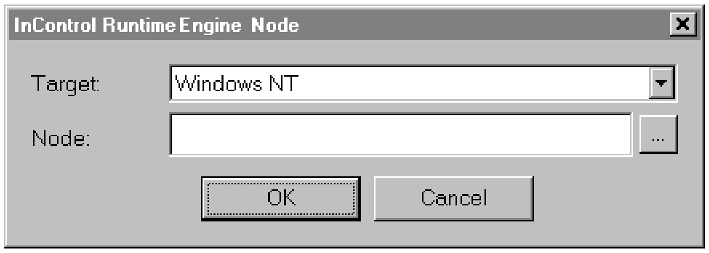
Using the Configure Runtime Engine Wizard

Place the Configure Runtime Engine wizard on an InTouch window to enable the operator to make configuration changes to the runtime engine on the specified node at runtime.



To specify the node for which the operator configures the runtime engine:

1. Double-click the wizard. The **InControl Runtime Engine Node** dialog box appears.



InControl Runtime Engine Node Dialog Box

2. Specify the target hardware platform.
3. To specify the runtime engine on the local node, leave the **Node** field blank.

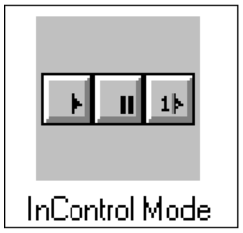
   To specify the runtime engine on a remote node, enter the node name in the **Node** field.

   When the operator clicks the wizard from the InTouch window at runtime, the **Online Runtime Engine Properties** dialog box appears.

For a complete description of the fields of the **Online Runtime Engine Properties** dialog box, see "Checking General Properties of the Runtime Engine" of the "InControl System Administration" chapter.

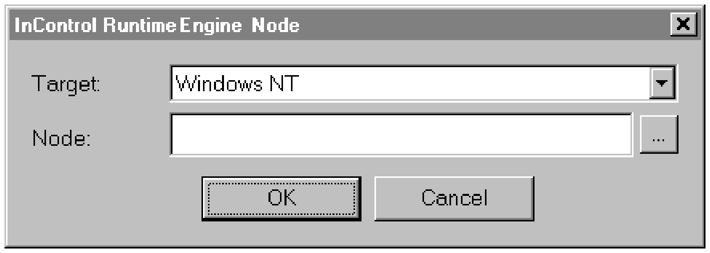
Using the InControl Mode Wizard

Place the InControl Mode wizard on an InTouch window to enable the operator to change the mode (Run Project, Pause, Single Scan) for the project that has been downloaded to the runtime engine.



To specify the node controlled by the InControl Mode Wizard:

1. Double-click the wizard. The **InControl Runtime Engine Node** dialog box appears.



InControl Runtime Engine Node Dialog Box

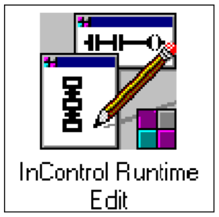
2. Specify the target hardware platform.
3. To specify the runtime engine on the local node, leave the **Node** field blank.

   To specify the runtime engine on a remote node, enter the node name in the **Node** field.

For more information about program mode, see "Selecting Runtime Options" of the "Running a Project" chapter.

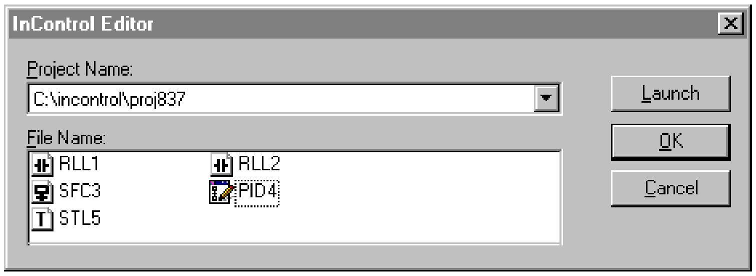
Using the InControl Runtime Edit Wizard

Place the InControl Runtime Edit wizard on an InTouch window to enable the operator to open an InControl program at a specific point in the program. After the editor opens the program, the operator can use all the functions of the InControl development environment to edit, compile, download, and run the program. If there are any other programs in the project, the operator can access these as well. The InControl development environment must be installed on the operator's hardware system before the operator can edit any project files.



When the program is opened, the editor displays the program at the last location of the cursor when the program was closed. If you want to direct the operator's attention to the code that controls a valve, for example, which is on an InTouch window, place the Edit wizard near the valve. Open the program, move the cursor to the valve code, and then close the program.

If you double-click the InControl Runtime Edit wizard from the InTouch WindowMaker, the **InControl Editor** dialog box appears.


InControl Editor

Project Name: C:\incontrol\proj837

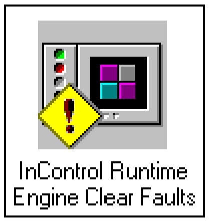
File Name: RLL1, RLL2, SFC3, PID4, STL5

InControl Editor Dialog Box

Button / Field	Description
Project Name	Select the InControl project containing the program to edit.
File Name	Select the program to be edited.
Launch	Click **Launch** to run the specified InControl program from WindowMaker.
OK	Accepts the selected program to be edited.
Cancel	Closes the wizard dialog box without saving entries.

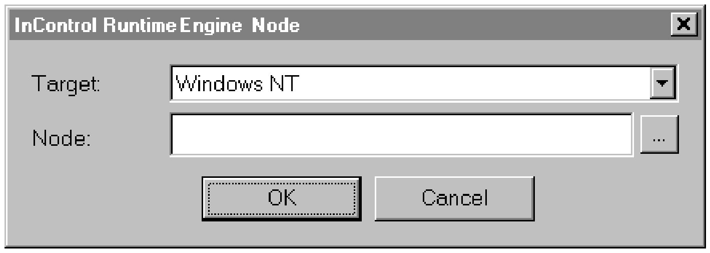
Using the InControl Clear Faults Wizard

Place the InControl Runtime Engine Clear Faults wizard on an InTouch window to enable the operator to clear any runtime engine faults on the specified node at runtime. When the operator clicks the wizard at runtime, runtime engine status bits, such as RTEngine.ScanOverrun, are cleared.



To specify the node for which the operator clears faults:

1. Double-click the wizard. The **InControl Runtime Engine Node** dialog box appears.



InControl Runtime Engine Node

Target: Windows NT

Node:

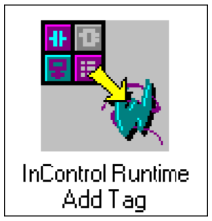
InControl Runtime Engine Node Dialog Box

2. Specify the target hardware platform.
3. To specify the runtime engine on the local node, leave the **Node** field blank.

   To specify the runtime engine on a remote node, enter the node name in the **Node** field.

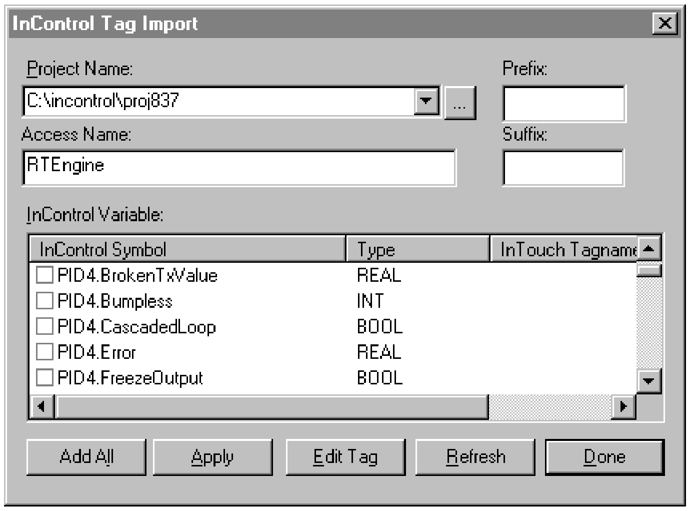
Using the InControl Runtime Add Tag Wizard

Use the InControl Runtime Add Tag wizard to link InTouch tags to symbols (variables) used in an InControl project. The InControl development environment must be installed where you are running InTouch for you to link InControl symbols to InTouch tags.



Displaying the InControl Symbols

When you click the InControl Runtime Add Tag wizard from WindowMaker, the **InControl Tag Import** dialog box appears. All symbols used in the programs within the project are listed. To link a symbol to a tag, see "Linking InControl Symbols to InTouch Tags."



Project Name: C:\incontrol\proj837
Prefix:
Access Name: RTEngine
Suffix:
InControl Variable:

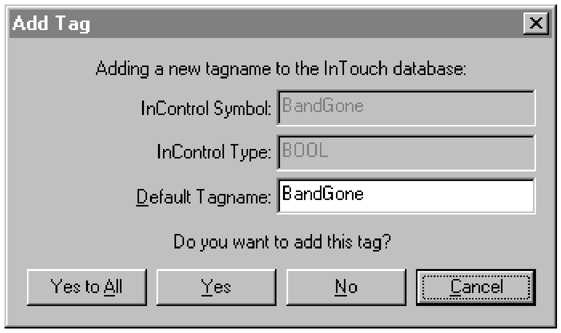
InControl Symbol	Type	InTouch Tagname
PID4.BrokenTxValue	REAL	
PID4.Bumpless	INT	
PID4.CascadedLoop	BOOL	
PID4.Error	REAL	
PID4.FreezeOutput	BOOL	

InControl Tag Import Dialog Box

Button / Field	Description
Project Name	Select the project with the symbols to be linked to tags. The InControl projects listed in this field are the same ones configured in the InControl Development environment, if it is installed on the local node. If InControl is not installed on the local node, this list is blank. You can browse the network and add InControl projects to the list.
Access Name	Specify the access name to be used for accessing tags you link. For the local node, use the default value RTEngine. For a remote node, enter the access name definition listed in the InTouch **Access Names** dialog box for the node (click **Access Names** on the **Special** menu to open this dialog box). For multiple remote nodes, the access names for the nodes must be unique. Define an access name for each remote node.
Prefix/Suffix	Optional. Enter a prefix or a suffix to the tag names that are imported into InTouch. This enables you to group tags, by project for example, if you are displaying several groups in one InTouch window.
InControl Symbol	Symbol name used in InControl. Double-click the symbol name to display the **Add Tag** dialog box. The **Add Tag** dialog box is described in "Linking InControl Symbols to InTouch Tags." If the tag has already been added to the InTouch database, double-click the symbol name to display the InTouch **Tagname Dictionary** dialog box.
Type	Read-only field displays the data type for the symbol.
InTouch Tagname	Tag name to be used in InTouch. Field is blank until the InControl symbol has been linked to an InTouch tag.
Add All	Links all symbols in the project to InTouch tags.
Apply	Links selected InControl symbols to InTouch tags.
Edit Tag	Displays the **Add Tag** dialog box if the tag has not been added to the InTouch database. Displays the InTouch **Tagname Dictionary** dialog box if the tag has already been added to the InTouch database.
Refresh	Redisplays all information in the dialog box.
Done	Closes the dialog box.

Linking InControl Symbols to InTouch Tags

When you display the **InControl Tag Import** dialog box, all symbols used in the programs within the project are listed. Select one or more symbols and click **Apply**. The **Add Tag** dialog box appears. You can also double-click an individual symbol to display the **Add Tag** dialog box.


Add Tag

Adding a new tagname to the InTouch database:

InControl Symbol: BandGone

InControl Type: BOOL

Default Tagname: BandGone

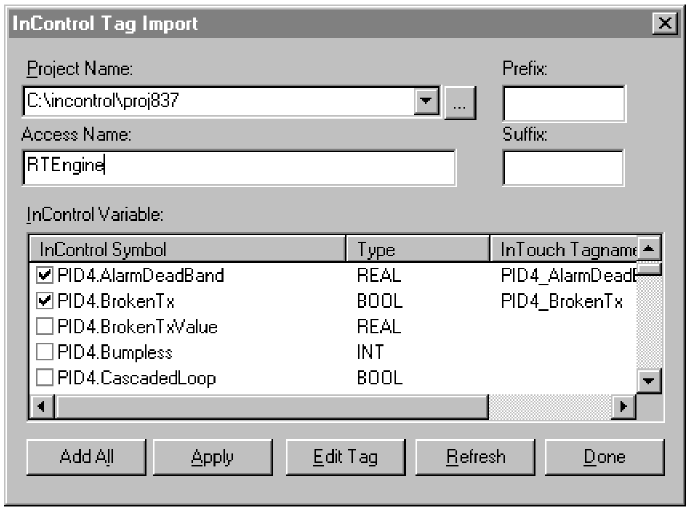
Do you want to add this tag?

Add Tag Dialog Box

Button / Field	Description
InControl Symbol	Read-only field displays the name of the symbol to be linked.
InControl Type	Read-only field displays data type of the symbol.
Default Tagname	Tag name to be used by InTouch. The InControl symbol name is the default.
Yes to All	Links all selected symbols to InTouch tags.
Yes	Links the selected symbol to an InTouch tag.
No	Closes the dialog box without linking the symbol to a tag. If multiple symbols are selected, the next symbol is displayed.
Cancel	Closes the dialog box without saving entries.

Example Showing Linked Symbols

The following figure shows the **InControl Tag Import** dialog box after two symbols have been linked to tags.



InControl Tag Import

Project Name: C:\incontrol\proj837

Prefix:

Access Name: RTEngine

Suffix:

InControl Variable:

InControl Symbol	Type	InTouch Tagnam
✓ PID4.AlarmDeadBand	REAL	PID4_AlarmDead
✓ PID4.BrokenTx	BOOL	PID4_BrokenTx
PID4.BrokenTxValue	REAL	
PID4.Bumpless	INT	
PID4.CascadedLoop	BOOL	

Linking Symbols and Tags

Linking Tags on Remote Systems

You can link InControl symbols, which are used in projects located on remote nodes, to InTouch tags. The remote system must contain a copy of InControl, and it must be networked to the system on which you are running InTouch. If you are using DDE, you must implement NetDDE between the two systems. If you are using SuiteLink, you are not required to use DDE.

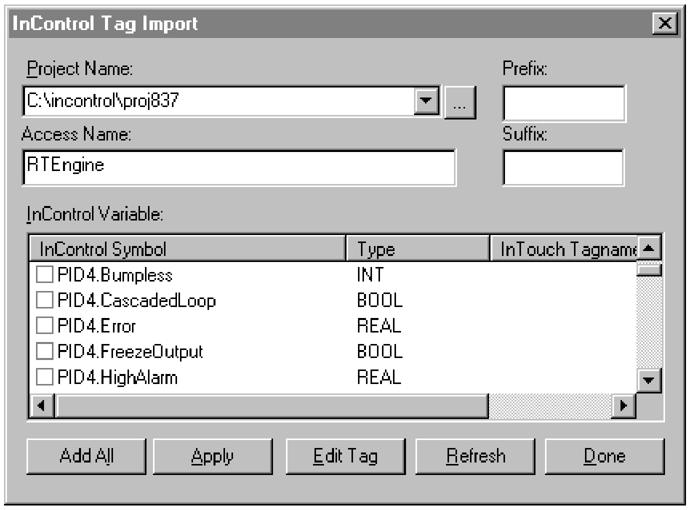
For information about implementing NetDDE, See the following sources:

- Technical Note #30 "Configuring Windows NT 4.0 DDE Shares for NetDDE." The *Tech Note* publication is published periodically by the Wonderware Technical Support group. E-mail your questions or requests to techpubs@wonderware.com.
- The Comprehensive Support Knowledge Base CD. Phone Wonderware Technical Support at (949) 727-3299 for more information about the Comprehensive Support Knowledge Base CD.
- NetDDE Extensions for Windows NT User's Guide, part number 05-139.

Before you can link the symbols in the project located on the remote node, you need to add that project to the Project List on the computer on which you are running InTouch. See "Adding a Project" of the "Project Organization and Management" chapter for a detailed procedure.

To link the symbols of a remote node to the InTouch tags on a local system:

1. Click the InControl Runtime Add Tag wizard from **Window Maker**, as described in "Using the InControl Runtime Add Tag Wizard", to display the **InControl Tag Import** dialog box.



InControl Tag Import

Project Name: C:\incontrol\proj837

Access Name: RTEngine

Prefix:

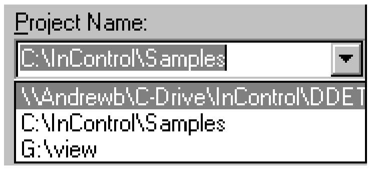
Suffix:

InControl Variable:

InControl Symbol	Type	InTouch Tagname
☐ PID4.Bumpless	INT	
☐ PID4.CascadedLoop	BOOL	
☐ PID4.Error	REAL	
☐ PID4.FreezeOutput	BOOL	
☐ PID4.HighAlarm	REAL	

InControl Tag Import Dialog Box

2. Click the display tool to the right of the **Project Name** field to display all the projects in the Project List.



Project Name: C:\InControl\Samples

\\Andrewb\C-Drive\InControl\DDET
C:\InControl\Samples
G:\view

3. Select the remote project. The **InControl Tag Import** dialog box displays the symbols for the remote project.

4. Link the symbols to the InTouch tags, described in "Linking InControl Symbols to InTouch Tags."

Using the InTouch Tag Browser

As an alternative to using the InControl Add Tag wizard for importing InControl symbols into InTouch, you can use the InTouch Tag Browser. The Tag Browser is especially useful when you are writing a script and need to reference an InControl symbol.

For detailed information about using the InTouch Tag Browser, see the *InTouch User's Guide*.

Note that you can also use the Import and Export tools on the Symbol Manager toolbar to exchange symbols between InControl and InTouch. The comma-separated format (CSV) is used, which means you can easily edit a file of symbol data using an ASCII text editor or Excel.

For more information about using the Import and Export tools, see "Transferring Symbol Databases" in the "Defining Variables" chapter.

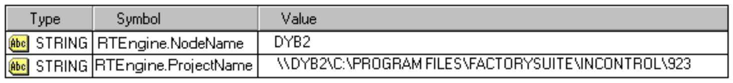
If you intend to use InTouch and InControl on separate systems and need to view the InControl symbols from InTouch, you must install the InControl Tag Browser files on the system where InTouch is located. Run the InControl setup program on the InTouch system and choose **InTouch Extensions** when prompted for the setup option.

For more information about installing InControl, see "Installation Guidelines" in the "Getting Started with InControl" chapter.

Project Node/Name and InTouch

You can use two runtime engine symbols (NodeName and ProjectName) to send a project name and the node on which the project is executing to an HMI, such as InTouch.

For example, to display the name of a project (called 923) and the node where it is executing (called DYB2), in the Watch Window, add the following symbols to the Watch Window: RTEngine.ProjectName and RTEngine.NodeName. The following figure shows the symbols as they appear in the Watch Window.



Type	Symbol	Value
STRING	RTEngine.NodeName	DYB2
STRING	RTEngine.ProjectName	\\DYB2\C:\PROGRAM FILES\FACTORYSUITE\INCONTROL\923

Displaying Project Information

InControl QuickScript Functions

InTouch scripting is one of the most powerful features of an InTouch application. The InTouch QuickScript capabilities allow the operator to execute commands and logical operations based on specified criteria. For example, you can define a button that an operator selects to open an InControl project or to clear faults in the runtime engine.

InControl supports two QuickScript functions that you can use within an InTouch script: InControl() and InControlRuntimeEngine().

InControl()

Opens an InControl project, program, I/O configuration, etc.

Syntax

```
InControl("Project","Command","Param");
```

Parameter	Description	Values
Project	Path of an InControl project.	<user-specified>
Command	Operation to be executed.	Launch
Param	Name of object to be started.	<user-specified>

Example

```
InControl("c:\Projects\SeamWeld","Launch","BandLogic");
```

opens the program called BandLogic of the project called SeamWeld in the InControl development environment.

InControlRuntimeEngine()

Sends a command to the runtime engine.

Syntax

`InControlRuntimeEngine(`*`"Nodename","Command","Param"`***`);`**

Parameter	Description	Values
Nodename	Name of the node where the runtime engine is executing.	<user-specified>
Command	Operation to be executed.	Mode Configure ClearFaults
Param	Used only with the Mode command; specifies mode of the runtime engine.	Run Pause Single Scan Stop

Examples

```
InControlRuntimeEngine("Node75","Mode","Stop");
```

stops the project that is running on the node called Node75.

```
InControlRuntimeEngine("Node42","Configure","");
```

displays the Online Runtime Engine Properties dialog box for the runtime engine that is running on the node called Node42.

Index

Glossary

Action

Named collection of operations associated with one or more Steps in an SFC.

Action Manager

Use the Action Manager to rename and delete SFC Actions.

Action Qualifier

Graphical programming element in an SFC associated with each Action block that controls execution of the action logic relative to the period during which the associated Step is active.

Active File

Program File that is contained in the top Program Editor window with its title bar highlighted. Commands that are executed from the menus or by clicking on buttons on the tool bars are performed on the active file.

ActiveX

See FOE.

Analog Alarm

InControl FOE that monitors analog input signals for alarm conditions.

ANY

The ANY data type is a generic data type. ANY can assume the type and range of any of the data types that are supported by InControl with these exceptions: File, TMR, and User-Defined.

ANY_BIT

Generic data type that can represent these data types: DWORD, WORD, BYTE, BOOL, including an individual bit within these data types.

ANY_DATE

Generic data type that can represent these data types: DT, DATE, TOD.

ANY_INT

Generic data type that can represent the INT data type.

ANY_NUM

Generic data type that can represent these data types: ANY_REAL and ANY_INT.

ANY_REAL

Generic type name that can represent the REAL data type.

Array

Set of data values, all of the same type. Individual elements can be referenced by an expression consisting of the array name and an indexing expression.

Board

See I/O Card.

Background Execution

Some functions and factory objects can run in background, which means the runtime engine scan is not delayed while the function or factory object completes execution.

BOOL

Member of the ANY_BIT group of data types. BOOL data types are valid in ANY InControl instruction or Function Block that accepts an ANY, ANY_BIT, or BOOL data type. A BOOL is one bit in length and can have one of two values: TRUE (1, or on) or FALSE (0, or off).

Boolean Transition

Type of Transition in an SFC represented by an expression consisting of variables and operators that evaluate to a single Boolean result. If the Boolean result is TRUE, then the SFC transition condition is satisfied.

Boolean

Logical element or expression that evaluates to either TRUE or FALSE.

BYTE

Member of the ANY_BIT group of data types. BYTE data types are valid in any InControl instruction or Function Block that accepts an ANY, ANY_BIT, or BYTE data type. A BYTE is an unsigned integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. A BYTE is 8 bits in length and has a range of 0 to 255.

Card

See I/O Card.

Child SFC

The Macro is a specialized POU that provides a means of including one SFC, the child SFC, for execution from a Step in another SFC, the Parent SFC.

Coil

RLL graphical programming element that represents a Boolean output Variable.

Connector

Group of I/O ports usually routed to one physical connector on a Board.

Contact

RLL graphical programming element that represents a Boolean input Variable.

Control Loop Element

See Loop.

DATE

Member of the ANY_DATE group of data types. DATE data types are valid in ANY InControl instruction or Function Block that accepts an ANY, ANY_DATE, or DATE data type.

DDE

See Dynamic Data Eschange.

DINT

Member of the ANY_NUM group of data types. DINT data types are valid in ANY instruction or Function Block that accepts an ANY, ANY_NUM, ANY_INT, or DINT data type. The DINT is a signed integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. The DINT is 32 bits in length and has a range of -2147483648 to +2147483647.

DT

Member of the ANY_DATE group of data types. DT data types are valid in ANY instruction or Function Block that accepts an ANY, ANY_DATE, or DT data type. Format: DT#YYYY-MM-DD-HH:MM:S.S YYYY (100-2100) = year MM (1-12) = month DD (1-31) = day of the month HH (0-23) = hour MM (0-59) = minute

S.S (0.0-59.0) = seconds (REAL number)

DWORD

Member of the ANY_BIT group of data types. DWORD data types are valid in ANY InControl instruction or Function Block that accepts an ANY, ANY_BIT, or DWORD data type. A DWORD is an unsigned integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. A DWORD is 32 bits in length and has a range of 0 to 4294967295.

Dynamic Data Exchange

DDE is the passage of data between applications, accomplished without user involvement or monitoring. In the Windows environment, DDE is achieved through a set of message types, recommended procedures (protocols) for processing these message types, and some newly defined data types. By following the protocols, applications that were written independently of each other can pass data between themselves without involvement on the part of the user, e.g. InTouch.

Factory Object

See FOE.

FactorySuite

FactorySuite 2000 is the world's first integrated, component-based MMI System. With FactorySuite 2000, you have access to all the information you need to run your factory. - visualization, optimization and control, plant floor data collection, and data storage and analysis -- to make your plant truly productive.

File

The FILE data type is a member of the ANY group of data types. FILE is a Structure that is designed only for the file control variables used with the RLL and Structured Text file functions.

FOE

InControl is compatible with the ActiveX Server specification. The InControl Factory Object (FOE) editor is an ActiveX container, which enables you to use ActiveX controls within an InControl Project. An ActiveX control must be installed within InControl before you can configure and run it. After installation, it is referred to as an InControl Factory Object (FOE). Like other InControl programs, an FOE can run independently. You can also call it for execution from another Program.

Force

Watch Window utility that sets a Variable to a value that does not change as the Program runs until you Unforce it. *See also Unforcee.*

Function

A POU consisting of a set of programming instructions that can be called for execution by another POU. Functions are algorithms that carry out a single operation, such as Square Root, Rotate Left, Tangent, etc. A true function returns a value and is used on the right side of an Assignment statement. *See also Procedure*.

Function Block

A POU consisting of a set of programming instructions that can be called for execution by another POU. Unlike a function, one or more instances of a function block type can be created; and local variables maintain their values between calls.

INT

Member of the ANY_NUM group of data types. INT data types are valid in ANY instruction or Function Block that accepts an ANY, ANY_NUM, ANY_INT, or INT data type. The INT is a signed integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. The INT is 16 bits in length and has a range of -32768 to +32767.

I/O Card

Plugs into one of the expansion slots of the InControl system unit and connects to the peripheral I/O racks and modules. Also called *Card* or *Board.*

Jump

Graphical programming element in an SFC or RLL Program that directs program flow to another location in the program identified by a Label.

Label

Graphical programming element in an SFC or RLL Program that identifies where program flow is to resume from its corresponding Jump.

Loop

Graphical programming element in an SFC diagram. The Loop Element contains two Transition conditions: one directs Program flow to continue in the downward direction, and the other directs program flow to loop back. Multiple Loop Elements can be nested within each other, but they can not cross each other and can not enter Select Diverges or Parallel Diverges.

LREAL

Member of the ANY_NUM group of data types. LREAL data types are valid in any instruction or Function Block that accepts an ANY, ANY_NUM, ANY_REAL, or LREAL data type. An LREAL number data type is a 64-bit value composed of one or more of the digits (0-9), is signed, and contains a decimal point. The range for LREAL numbers is the following: -1.79769313486231 E308 (negative) to +1.79769313486231 E308 (positive), and includes zero.

Macro

A specialized POU that provides a means of including one SFC, the child, for execution from a Step in another SFC, the parent. The Macro Step is the graphical element in an SFC that represents the inclusion of another entire SFC as a single Step. The included SFC begins execution at its Start Step when the Macro Step that calls it becomes active. Execution in the Macro Step is completed when the included SFC reaches its End Step.

OCX

See FOE.

Parallel Divergence

SFC graphical programming element that splits a control path into two or more parallel paths. When Program execution reaches the beginning of a Parallel Divergence, all the subsequent control paths become active in parallel. These control paths continue to be active until all the control paths within a Parallel Diverge reach the point of convergence. At this point, all the paths within the Parallel Divergence are deactivated and the control path below the convergence point will become active.

Parent SFC

The Macro is a specialized POU that provides a means of including one SFC, the Child SFC, for execution from a Step in another SFC, the parent SFC.

PID Loop

InControl FOE that manages proportional, integral, and derivative feedback control.

Port

I/O port usually consisting of eight I/O bits. A port may be accessed as an integer within programs.

POU

The POU (Program organization unit) is defined by the IEC 61131-3 standard as the basic programming unit. InControl supports these POUs: program, Function, Function Block, and Macro.

Procedure

Predefined algorithm that carries out a single operation, such as Delete File, Abort All, Rewind File, etc. A procedure does not return a value. *See also Function*.

Program

A POU consisting of a block of code that can be scheduled to execute automatically every scan.

Project

Organizes or groups the application programs and configuration files for an application in a separate subdirectory.

REAL

Member of the ANY_NUM group of data types. REAL data types are valid in ANY instruction or Function Block that accepts an ANY, ANY_NUM, ANY_REAL, or REAL data type. A REAL number data type is a 32-bit value composed of one or more of the digits (0-9), is signed, and contains a decimal point. The range for REAL numbers is the following: -3.402823 E38 (negative), to +3.402823 E38 (positive), and includes zero.

Relay Ladder Logic

RLL is the graphical programming language used for describing application Program logic based on an electrical relay Contact and Coil analogy.

Retentive Variable

A Variable that retains its value in the event of a power loss.

RLL Transition

Transition in an SFC that is programmed using Relay Ladder Logic. The RLL transition consists of a single RLL rung with an output Coil that has the same name as the RLL transition. When this output coil has power flow, the SFC transition condition is satisfied.

Runtime Engine

Module responsible for scheduling and executing the Program logic associated with the Project's source code, e.g., SFC, RLL, etc. This module also performs as a SuiteLink or DDE server to SuiteLink or DDE clients such, as InTouch.

Select Divergence

SFC graphical programming element that splits a single control path into two or more paths. The control path selected for execution is determined by the Transition conditions that are located at the beginning of each of the new control paths.

Sequential Function Chart

SFC is the graphical programming language for diagramming sequential logic using Steps, Transitions and Actions.

SFC Transition Coil

SFC graphical programming element that can be associated with a Step or a Macro Step and provide Program flow control. Associated with a Step, the SFC Transition Coil can abort the Step and direct program flow to another Step. Associated with a Macro Step, the SFC transition coil can abort the Child SFC and direct program flow to another Step in the Parent SFC.

SINT

Member of the ANY_NUM group of data types. INT data types are valid in ANY instruction or Function Block that accepts an ANY, ANY_NUM, ANY_INT, or SINT data type. The SINT (short integer) is a signed integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. The INT is 8 bits in length and has a range of -128 to +127.

Step

Graphical element in an SFC that represents a state or span of TIME in the Program execution during which the actions and functions associated with the Step are performed.

STL

See Structured Text Language.

String

Member of the ANY group of data types. STRING data types are valid in any InControl instruction or Function Block that accepts an ANY or STRING data type. The format for a STRING data type consists of a string of up to 1024 ASCII characters in single quotation marks.

Structure

See UUSER_DEFINED.

Structured Text Language

The Structured Text programming language is a subset of an IEC61131 compliant set of text-based instructions.

SuiteLink

The SuiteLink driver provides reliable, high-speed data collection for controllers, PC boards and other devices. Unlike DDE, which transfers only one message at a TIME, SuiteLink transfers blocks of messages, allowing faster and more efficient data transfer.

Symbol

See Variable.

Tagname

The name assigned to a Variable in the InTouch database to represent an InControl variable.

Task View

Graphical display of the programs in a Project, arranged by priority of execution.

TIME

Member of the ANY group of data types. TIME data types are valid in any InControl instruction or Function Block that accepts an ANY or TIME data type. The format of the TIME data type consist of a T# or t# followed by a sequence of one or more numbers and time unit specifiers. Examples: T#1D2h = 1 day and 2 hours t#26H = 26 hours t#5m45s = 5 minutes and 45 seconds t#26S200MS = 26 seconds and 200 milliseconds T#900ms = 900 milliseconds

TMR

The TMR data type is a member of the ANY group of data types. TMR data types are valid in any instruction or Function Block that accepts an ANY or TMR data type.

TOD

The TOD data type is a member of the ANY_DATE group of data types. TOD data types are valid in ANY instruction or Function Block that accepts an ANY, ANY_DATE, or TOD data type.

Transition

Graphical element in an SFC that evaluates to a Boolean result. This Boolean result determines when Program flow is passed from Step(s) preceding the transition to Step(s) following the transition.

UDINT

The UDINT data type is equivalent to the DWORD data type. An InControl enhancement to the ANY_BIT data types makes the UDINT (unsigned double integer) data type unnecessary.

UINT

The UINT data type is equivalent to the WORD data type. An InControl enhancement to the ANY_BIT data types makes the UINT (unsigned integer) data type unnecessary.

Unforce

Watch Window utility that removes the Force status from a Variable, allowing the Program to write values to it. *See also Unforce*.

USER_DEFINED

Member of the ANY group of data types. The USER-DEFINED data type can be either a Structure of a enumeration. A structure consists of a group of data types (integers, Booleans, strings, etc.) that Function as a group. They do not all have to be same data type. An enumeration is a type of structure, the members of which are a set of DINT data types. Use an enumeration when you need to define a group of named constants. USER-DEFINED data types are valid in any InControl instruction or Function Block that accepts an ANY or USER-DEFINED data type.

USINT

The USINT data type is equivalent to the BYTE data type. An InControl enhancement to the ANY_BIT data types makes the USINT (unsigned short integer) data type unnecessary.

Variable

Internal memory location that contains Project data. The content of the information is defined by the data type and can be REAL numbers, integers, strings of characters, etc. The Symbol Manager is used to define a variable and to assign it a symbolic name and data type. Also called *symbol*.

WORD

Member of the ANY_BIT group of data types. WORD data types are valid in ANY InControl instruction or Function Block that accepts an ANY, ANY_BIT, or WORD data type. A WORD is an unsigned integer data type that is composed of one or more of the digits (0-9) and cannot contain a decimal point. A WORD is 16 bits in length and has a range of 0 to 65535.