Logix 5000 Controllers Function Block Diagram

1756 ControlLogix, 1756 GuardLogix, 1769 CompactLogix, 1769 Compact GuardLogix, 1789 SoftLogix, 5069 CompactLogix, 5069 Compact GuardLogix, Studio 5000 Logix Emulate

Rockwell Automation

Programming Manual

Original Instructions

Important User Information

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

WARNING: Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.

ATTENTION: Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence.

IMPORTANT Identifies information that is critical for successful application and understanding of the product.

Labels may also be on or inside the equipment to provide specific precautions.

SHOCK HAZARD: Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.

BURN HAZARD: Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.

ARC FLASH HAZARD: Labels may be on or inside the equipment, for example, a motor control center, to alert people to potential Arc Flash. Arc Flash will cause severe injury or death. Wear proper Personal Protective Equipment (PPE). Follow ALL Regulatory requirements for safe work practices and for Personal Protective Equipment (PPE).

This manual includes new and updated information. Use these reference tables to locate changed information.

Grammatical and editorial style changes are not included in this summary.

Global changes

None in this release.

New or enhanced features

This table contains a list of topics changed in this version, the reason for the change, and a link to the topic that contains the changed information.

Change	Topic
Updated Legal notices.	Legal notices on page 8
Updated branding.	Throughout

This manual shows how to program Logix 5000 controllers with the function block diagram (FBD) programming language. This manual is one of a set of related manuals that show common procedures for programming and operating Logix 5000 controllers.

For a complete list of common procedures manuals, refer to the Logix 5000 Controllers Common Procedures Programming Manual, publication 1756-PM001.

The term Logix 5000 controller refers to any controller based on the Logix 5000 operating system.

Studio 5000 environment

The Studio 5000 Automation Engineering & Design Environment® combines engineering and design elements into a common environment. The first element is the Studio 5000 Logix Designer® application. The Logix Designer application is the rebranding of RSLogix 5000® software and will continue to be the product to program Logix 5000™ controllers for discrete, process, batch, motion, safety, and drive-based solutions.

The Studio 5000® environment is the foundation for the future of Rockwell Automation® engineering design tools and capabilities. The Studio 5000 environment is the one place for design engineers to develop all elements of their control system.

Additional resources

These documents contain additional information concerning related Rockwell Automation products.

Resource	Description
Industrial Automation Wiring and Grounding Guidelines, publication 1770-4.1	Provides general guidelines for installing a Rockwell Automation industrial system.
Product Certifications webpage, available at http://ab.rockwellautomation.com	Provides declarations of conformity, certificates, and other certification details.

View or download publications at http://www.rockwellautomation.com/literature. To order paper copies of technical documentation, contact the local Rockwell Automation distributor or sales representative.

Legal notices

Rockwell Automation publishes legal notices, such as privacy policies, license agreements, trademark disclosures, and other terms and conditions on the Legal Notices page of the Rockwell Automation website.

End User License Agreement (EULA)

You can view the Rockwell Automation End User License Agreement (EULA) by opening the license.rtf file located in your product's install folder on your hard drive.

The default location of this file is:

C:\Program Files (x86)\Common Files\Rockwell\license.rtf.

Open Source Software Licenses

The software included in this product contains copyrighted software that is licensed under one or more open source licenses.

You can view a full list of all open source software used in this product and their corresponding licenses by opening the oss_license.txt file located your product's OPENSOURCE folder on your hard drive. This file is divided into these sections:

- Components
  Includes the name of the open source component, its version number, and the type of license.
- Copyright Text
  Includes the name of the open source component, its version number, and the copyright declaration.
- Licenses
  Includes the name of the license, the list of open source components citing the license, and the terms of the license.

The default location of this file is:

C:\Program Files (x86)\Common Files\Rockwell\Help\<*product name*>\Release Notes\OPENSOURCE\oss_licenses.txt.

You may obtain Corresponding Source code for open source packages included in this product from their respective project web site(s). Alternatively, you may obtain complete Corresponding Source code by contacting Rockwell Automation via the **Contact** form on the Rockwell Automation website: http://www.rockwellautomation.com/global/about-us/contact/contact.page. Please include "Open Source" as part of the request text.

Program a Function Block Diagram

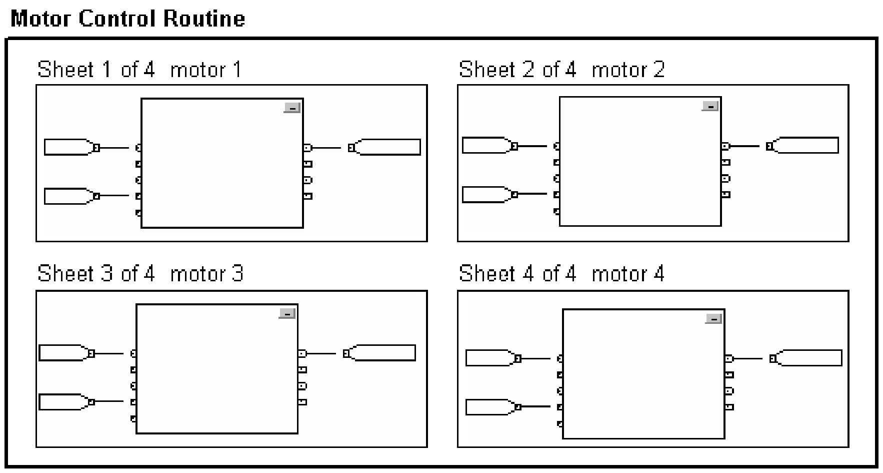
Introduction

To make it easier to navigate through a function block routine, divide the routine into a series of sheets.

- Sheets help organize function blocks and make them easier to locate. They do not affect the order in which the function blocks execute.
- When the routine executes, all the sheets execute.
- In general, use one sheet for each device, such as a motor or valve.

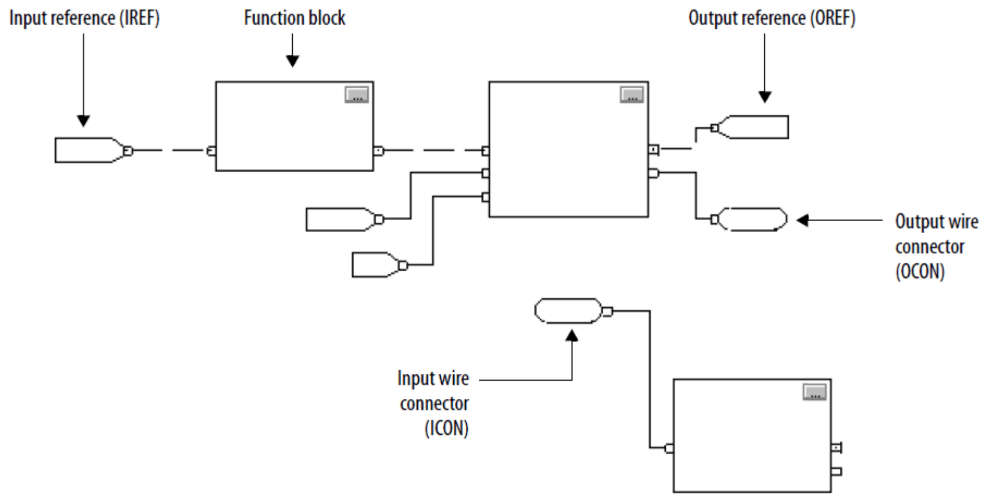
Example

Function block routine divided into sheets:



Choose the function block elements

Use these elements to control a device:



Guidelines to choose the function block elements:

To:	Use this element:
Supply a value from an input device or tag	Input reference (IREF)
Send a value to an output device or tag	Output reference (OREF)
Perform an operation on an input value or values and produce an output value or values	Function block or function Functions are similar to function blocks, but do not require backing tags, require less memory than function blocks, sometimes execute more quickly, and use less space in a function block diagram. **Tip:** Functions are available only in Logix Designer versions 32 and later on CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers.
Transfer data between function blocks when they are: • Far apart on the same sheet. • On different sheets within the same routine.	Output wire connector (OCON) and an input wire connector (ICON)
Disperse data to several points in the routine	Single output wire connector (OCON) and multiple input wire connectors (ICON)

Choose a tag name for an element

Each function block uses a tag to store configuration and status information about the instruction.

- The Logix Designer application automatically creates a tag for each new function block instruction. Use this tag, rename the tag, or assign a different tag.
- For IREFs and OREFs, create a tag or assign an existing tag.

For a:	Specify:
Tag	*tag_name*
Bit number of a larger data type	*tag_name.bit_number*
Member of a structure	*tag_name.member_name*
Element of a one dimension array	*tag_name[x]*
Element of a two dimension array	*tag_name[x,y]*

For a:	Specify:
Element of a three dimension array	*tag_name[x,y,z]*
Element of an array within a structure	*tag_name.member_name[x]*
Member of an element of an array	*tag_name[x,y,z].member_name*

where:

x is the location of the element in the first dimension.

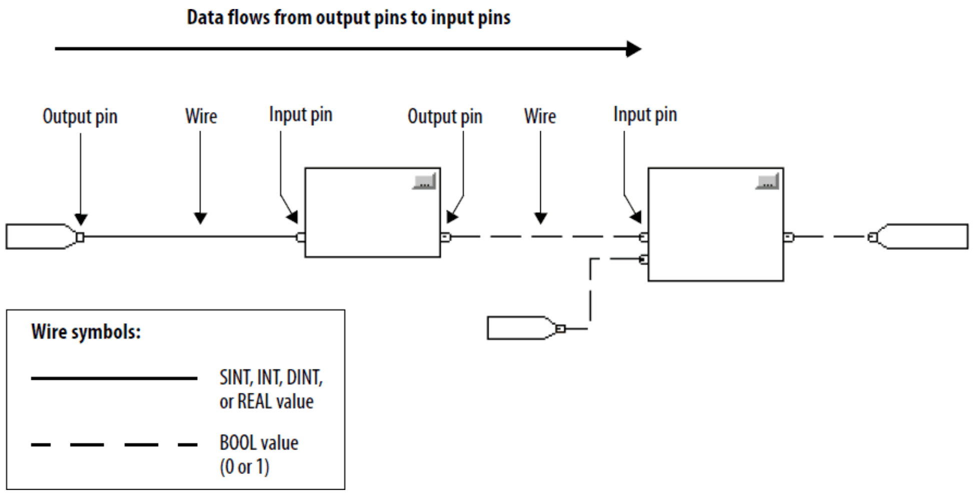
y is the location of the element in the second dimension.

z is the location of the element in the third dimension.

For a structure within a structure, add an additional *member_name*.

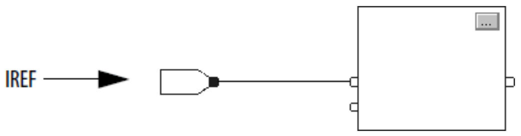
Define the order of execution

Define execution order (flow of data) by wiring elements together and indicating any input (feedback) wires, if necessary. The location of a block does not affect the order in which the blocks execute.

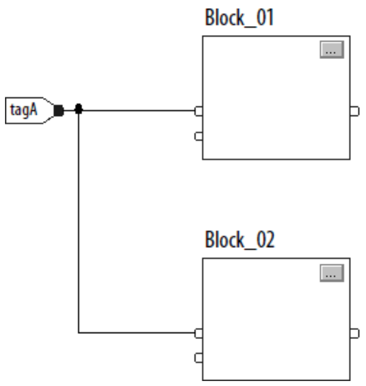


Data latching

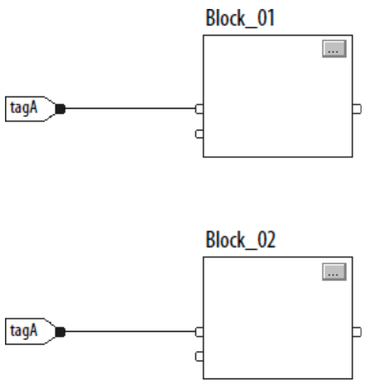
When an IREF specifies input data for a function block instruction, the data in that IREF is latched for the scan of the function block routine. The IREF latches data from program-scoped and controller-scoped tags. The controller updates all IREF data at the beginning of each scan as shown in this diagram.



In this example, the value of tagA is stored at the beginning of the execution of the routine. The stored value is used when Block_01 executes. The same stored value is also used when Block_02 executes. If the value of tagA changes during execution of the routine, the stored value of tagA in the IREF does not change until the next execution of the routine.

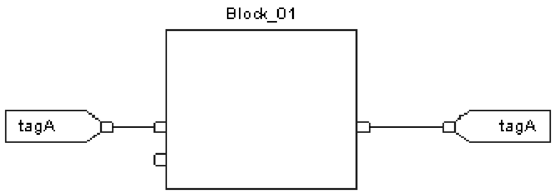


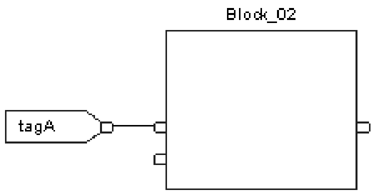
This example is the same as the previous example. The value of tagA is stored only once at the beginning of the execution of the routine. The routine uses this stored value throughout the routine.



With version 11 and later of the application, use the same tag in multiple IREFs and an OREF in the same routine. Because the values of tags in IREFs are latched every scan through the routine, all IREFs use the same value, even if an OREF obtains a different tag value during execution of the routine. In this example, if tagA has a value of 25.4 when the routine starts executing this scan, and Block_01 changes the value of tagA to 50.9, the second IREF wired into Block_02 still uses a value of 25.4 when Block_02 executes this scan.

The new tagA value of 50.9 is not used by any IREFs in this routine until the start of the next scan.





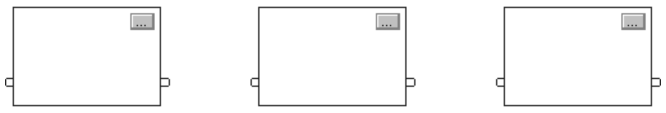
Order of execution

The Logix Designer application automatically determines the order of execution for the function blocks in a routine when:

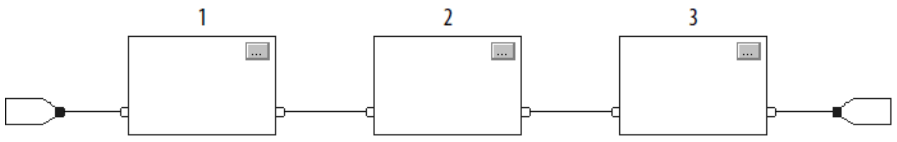
- Verifying a function block routine.
- Verifying a project that contains a function block routine.
- Downloading a project that contains a function block routine.

Define the order of execution by wiring function blocks together and indicating the data flow of any feedback wires, if necessary.

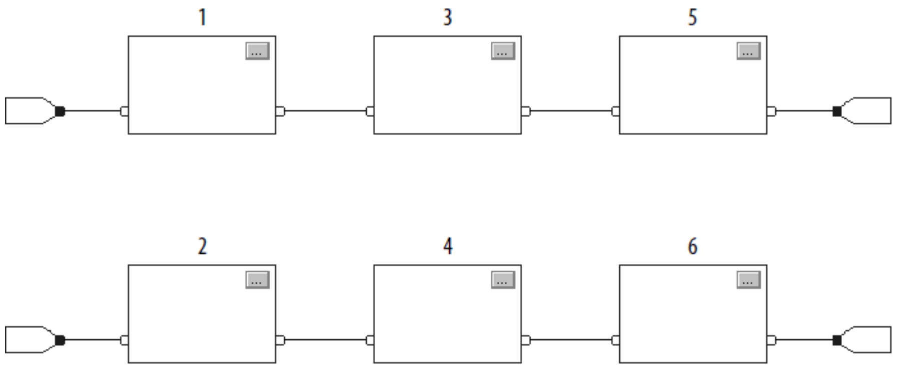
If function blocks are not wired together, it does not matter which block executes first. There is no data flow between the blocks.



If the blocks are wired sequentially, the order of execution moves from input to output. The inputs of a block require data to be available before the controller can execute that block. In this example, block 2 has to execute before block 3 because the outputs of block 2 feed the inputs of block 3.

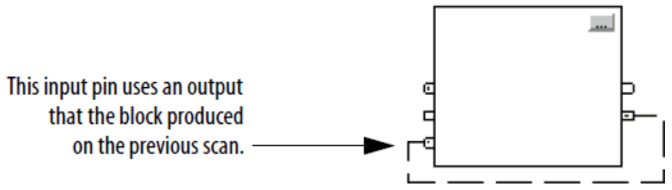


The order of execution is only relative to the blocks that are wired together. The two groups of blocks in this example are not wired together. The blocks within a specific group execute in the appropriate order in relation to the blocks in that group.

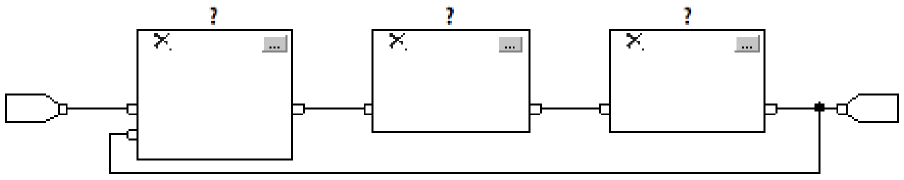


Resolve a loop

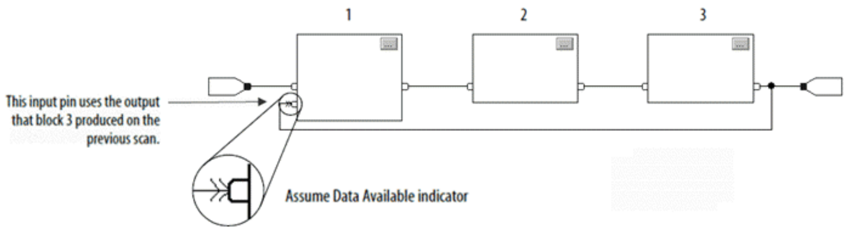
To create a feedback loop around a block, wire an output pin of the block to an input pin of the same block. In this example, the loop contains only a single block, so execution order does not matter.


This input pin uses an output that the block produced on the previous scan.

If a group of blocks are in a loop, the controller cannot determine which block to execute first, and it cannot resolve the loop, as illustrated in this example.

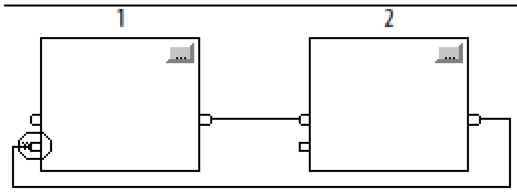
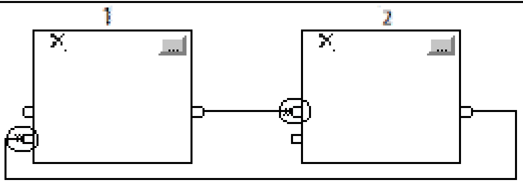


To identify which block to execute first, mark the input wire that creates the loop (the feedback wire) with the **Assume Data Available** indicator. In this example, block 1 uses the output from block 3 that was produced in the previous execution of the routine.


1
2
3
This input pin uses the output that block 3 produced on the previous scan.
Assume Data Available indicator

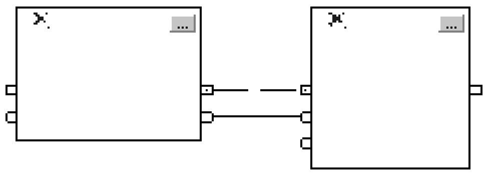
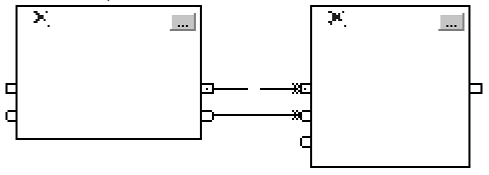
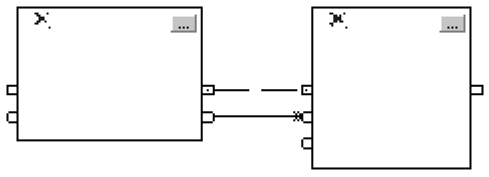
The **Assume Data Available** indicator defines the data flow within the loop. The arrow indicates that the data serves as input to the first block in the loop.

Do not mark all the wires of a loop with the **Assume Data Available** indicator.

This example is OK.	This example is not OK.
 The **Assume Data Available** indicator defines the data flow within the loop.	 The controller cannot resolve the loop because all the wires use the **Assume Data Available** indicator.

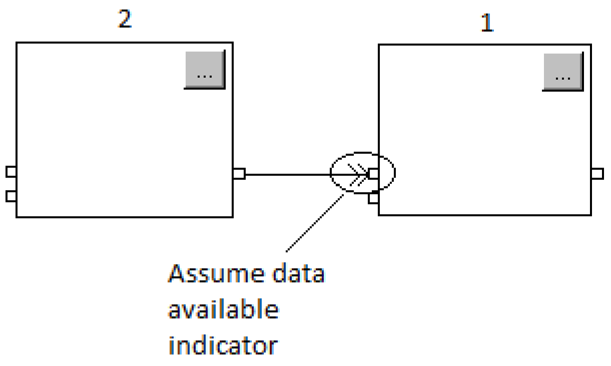
Resolve data flow between two blocks

When using two or more wires to connect two blocks, use the same data flow indicators for all of the wires between the two blocks.

This is OK	This is NOT OK
In this example, neither wire uses the **Assume Data Available** indicator.  In this example, both wires use the **Assume Data Available** indicator. 	In this example, only one wire uses the **Assume Data Available** indicator. 

Create a one scan delay

Use the **Assume Data Available** indicator to produce a one scan delay between blocks. In this example, block 1 executes first. It uses the output from block 2 that was produced in the previous scan of the routine.



Summary

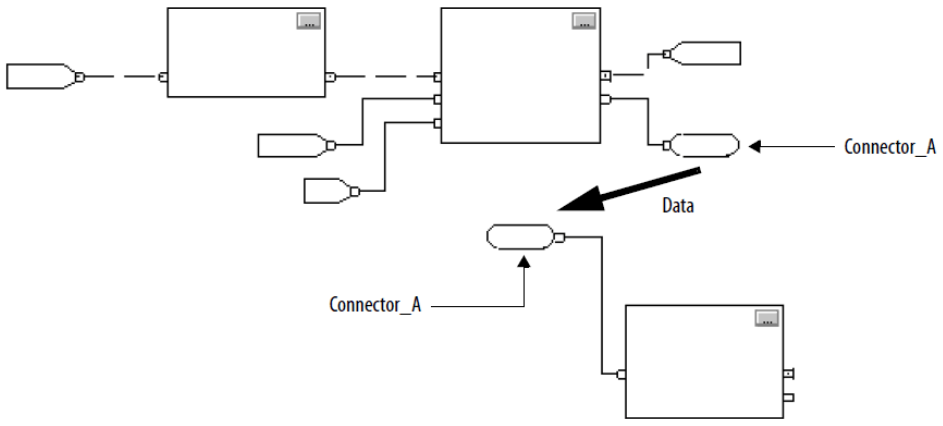
A function block routine executes in this order.

1. The controller latches all data values in IREFs.
2. The controller executes the other function blocks in the order determined by how they are wired.
3. The controller writes outputs in OREFs.

Identify any connectors

Like wires, connectors transfer data from output pins to input pins. Use connectors when:

- The elements to connect are on different sheets within the same routine.
- A wire is difficult to route around other wires or elements.
- Data should be dispersed to several points in the routine.



To use connectors, use these rules.

- Each OCON requires a unique name.
- Each OCON must have at least one corresponding ICON, such as an ICON with the same name as the OCON.
- Multiple ICONs can reference the same OCON. This allows dispersal of data to several points in a routine.

Define program/ operator control

These instructions support the concept of Program/Operator control.

- Enhanced Select (ESEL)
- Totalizer (TOT)
- Enhanced PID (PIDE)
- Ramp/Soak (RMPS)
- Discrete 2-State Device (D2SD)
- Discrete 3-State Device (D3SD)

Program/Operator control allows simultaneous control of these instructions from the user program and from an operator interface device. When in Program control, the instruction is controlled by the Program inputs to the instruction. When in Operator control, the instruction is controlled by the Operator inputs to the instruction.

Program or Operator control is determined by using these inputs:

Input	Description
.ProgProgReq	A program request to go to Program control.

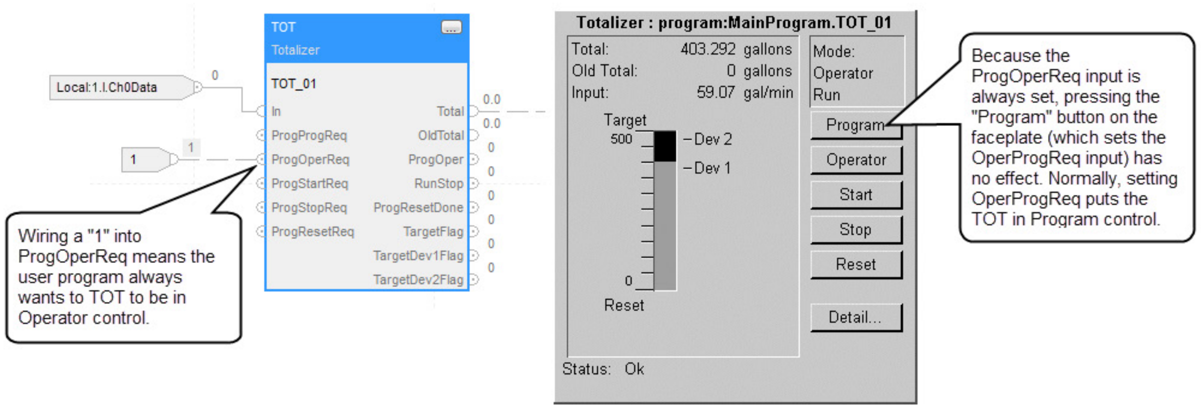
Input	Description
.ProgOperReq	A program request to go to Operator control.
.OperProgReq	An operator request to go to Program control.
.OperOperReq	An operator request to go to Operator control.

To determine whether an instruction is in Program or Operator control, examine the **ProgOper** output. If **ProgOper** bit is set, the instruction is in Program control. If **ProgOper** bit is cleared, the instruction is in Operator control.

Operator control takes precedence over Program control if both input request bits are set. For example, if **ProgProgReq** and **ProgOperReq** bits are set, the instruction goes to Operator control.

The Program request inputs take precedence over the Operator request inputs. This provides the capability to use the **ProgProgReq** and **ProgOperReq** inputs to *lock* an instruction in a desired control.

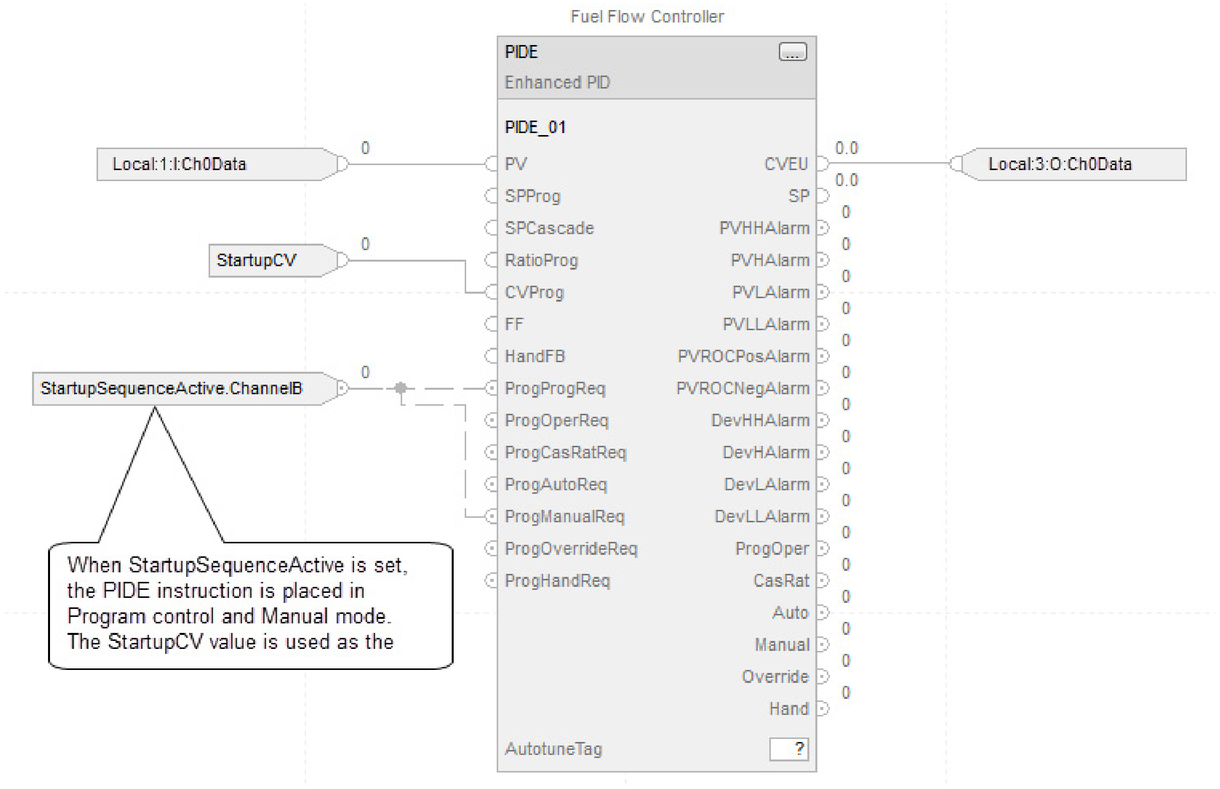
For example, assume that a Totalizer instruction is always used in Operator control, and the user program never controls the running or stopping of the Totalizer. In this case, wire a literal value of 1 into the **ProgOperReq** input. This prevents the operator from ever putting the Totalizer into Program control by setting the **OperProgReq** input from an operator interface device.



Wiring a "1" into ProgOperReq means the user program always wants to TOT to be in Operator control.

Because the ProgOperReq input is always set, pressing the "Program" button on the faceplate (which sets the OperProgReq input) has no effect. Normally, setting OperProgReq puts the TOT in Program control.

Likewise, constantly setting the **ProgProgReq** input can "lock" the instruction into Program control. This is useful for automatic startup sequences when the program should control the action of the instruction without an operator inadvertently taking control of the instruction. In this example, the program sets the **ProgProgReq** input during the startup, and clears the **ProgProgReq** input once the startup is complete. Once the **ProgProgReq** input is cleared, the instruction remains in Program control until it receives a request to change. For example, the operator sets the **OperOperReq** input from a faceplate to take control of that instruction. This example shows how to lock an instruction into Program control.



Operator request inputs to an instruction are always cleared by the instruction when it executes. This lets operator interface work with these instructions by setting the desired mode request bit. It is not necessary to program the operator interface to reset the request bits. For example, if an operator interface sets the **OperAutoReq** input to a PIDE instruction, when the PIDE instruction executes, it determines the appropriate response and clears the **OperAutoReq**.

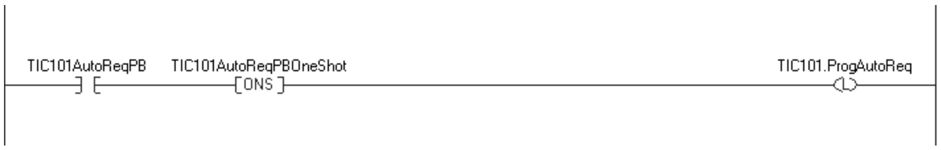
Program request inputs are not normally cleared by the instruction because these are normally wired as inputs into the instruction. If the instruction clears these inputs, the input would get set again by the wired input. In some situations, other logic should set the Program requests to be cleared by the instruction. In this case, set the **ProgValueReset** input and the instruction always clears the Program mode request inputs when it executes.

In this example, a rung of ladder logic in another routine is used to one-shot latch a **ProgAutoReq** input to a PIDE instruction when a push button is pushed. Because the PIDE instruction automatically clears the Program mode requests, no ladder logic is required to clear the **ProgAutoReq** input after the routine executes. The PIDE instruction receives only one request to go to Auto every time the push button is pressed.

Example

When the TIC101AutoReq button is pressed, one-shot latch **ProgAutoReq** for the PIDE instruction TIC101.

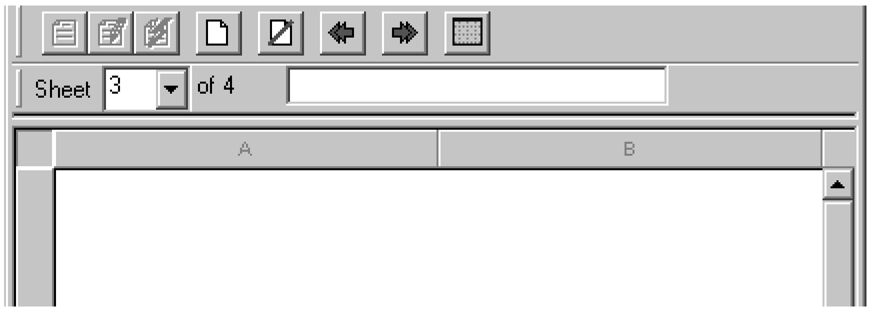
TIC101 is configured with the **ProgValueReset** input set, so when the PIDE instruction executes, it automatically clears **ProgAutoReq**.



Add a sheet

To add a sheet to a function block routine:

1. On the **Sheet** toolbar, click **Add Sheet** .

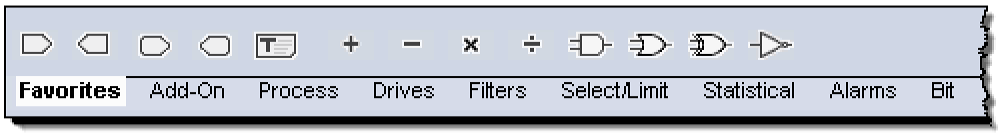


2. In the **Description** box, type a description of the sheet. Follow the IEC-1131 naming standard. The description must not be greater than 50 characters.

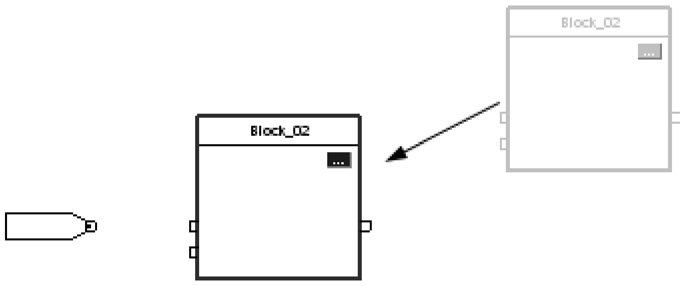
Add a function block element

Use the Language Element toolbar to add a function block element to a routine.

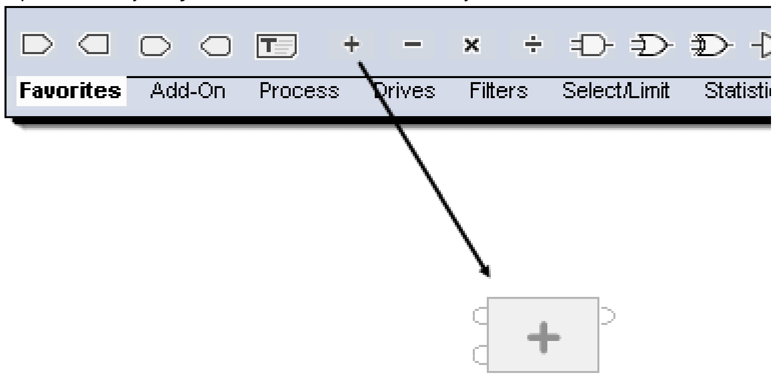
To add a function block element



1. On the **Language Element** toolbar, click the element to add.
2. On the function block diagram, drag the element to the desired location.



Tip: Alternatively, drag the button for the element directly to the desired location.



IMPORTANT Use caution when copying and pasting components between different versions of the Logix Designer application. The application only supports pasting to the same version or newer version. Pasting to a prior version of the application is not supported. When pasting to a prior version, the paste action may succeed but the results may not be as intended.

Use functions

Use Function Block functions to carry out mathematical, comparison, and logical operations. Functions in the **Function Block Diagram Editor** are similar to instructions, but do not require backing tags, require less memory than instructions, sometimes execute more quickly, and use less space in a function block diagram.

Tip: Function Block functions are available only on CompactLogix 5380, CompactLogix 5480, ControlLogix 5580, Compact GuardLogix 5380, and GuardLogix 5580 controllers.

This table lists the functions that are available in the categories on the **FBD Element toolbar**:

Toolbar category	Function	Symbol
Advanced Math	LN (Natural Log)	LN_f
	LOG (Log Base 10)	LOG_f
	XPY (X to the power of Y)	$x^{y}{}_f$
Compute / Math	ADD	+
	SUB	−
	MUL	×
	DIV	÷
	MOD (Modulo)	%
	SQR (Square Root)	$\sqrt{x}$
	NEG (Negate)	$-x$
	ABS (Absolute Value)	$\lvert x \rvert$
Compare	EQU (Equal To)	=
	GEQ (Greater Than or Equal To)	≥
	GRT (Greater Than)	>
	LEQ (Less Than or Equal To)	≤
	LES (Less Than)	<

Toolbar category	Function	Symbol
	LIM (Limit)	LIM_F
	MEQ (Mask Equal To)	MEQ_F
	NEQ (Not Equal To)	≠
Math Conversions	DEG (Radians to Degrees)	DEG_f
	RAD (Degrees to Radians)	RAD_f
	TRN (Truncate)	TRN_f
Move / Boolean Logical	AND (Bitwise And)	AND_f
	BAND (Boolean AND)	
	BXOR (Boolean Exclusive OR)	
	BNOT (Boolean NOT)	
	BOR (Boolean OR)	
	NOT (Bitwise Not)	NOT_f
	OR (Bitwise Inclusive Or)	OR_f
	XOR (Bitwise Exclusive Or)	XOR_f
Trigonometry	ACS (Arc Cosine)	ACS_f
	ASN (Arc Sine)	ASN_f
	ATN (Arc Tangent)	ATN_f
	COS (Cosine)	COS_f
	SIN (Sine)	SIN_f
	TAN (Tangent)	TAN_f

See also

FBD Element toolbar

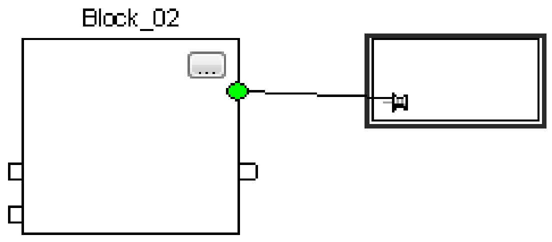
Function block display

Create a text box

Create a text box to add notes that clarify the function of an FBD element, such as blocks, references, and connectors. Text boxes are only stored in the offline, ACD project file. Text boxes are not downloaded into controller memory.

1. On the **Language Element** toolbar, click **Text Box** .
2. In the the FBD editor, drag the text box to a location near the element to which it applies.
3. Double-click the **text box** and enter the desired text.
4. Press **Ctrl+Enter**.

5. To attach the text box to a specific element, click the pin symbol in the text box and click the corresponding element. A green dot shows a valid connection point.



Language switching

Versions 17 and later of the Logix Designer application can display project documentation, such as tag descriptions and rung comments, for any supported localized language. Store project documentation for multiple languages in a single project file rather than in language-specific project files. Define all the localized languages that the project will support and set the current, default, and optional custom localized language. The default language is used if the current content of the language is blank for a particular component of the project. Use a custom language to tailor documentation to a specific type of project file user.

Enter the localized descriptions in a project by programming in that language, or by using the import/export utility to translate the documentation offline and import it back into the project. When language switching is enabled, users can dynamically switch between languages.

Project documentation that supports multiple translations within a project includes:

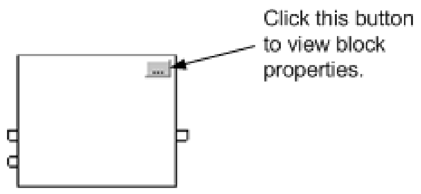
- Component descriptions in tags, routines, programs, user-defined data types, and Add-On Instructions.
- Equipment phases.
- Trends.
- Controllers.
- Alarm Messages (in ALARM_ANALOG and ALARM_DIGITAL configuration).
- Tasks.
- Property descriptions for modules in the Controller Organizer.
- Rung comments, SFC text boxes, and FBD text boxes.

For more information on enabling a project to support multiple translations of project documentation, see the online help.

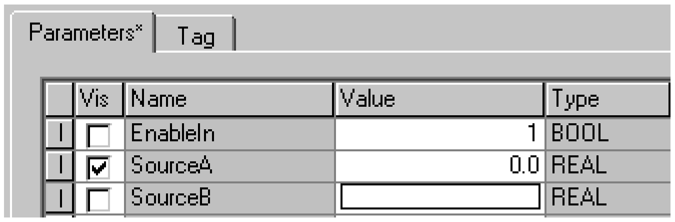
Connect elements

Show or hide a pin

Function Block instructions provide a set of pins for the default parameters. The rest of the pins are hidden. Show or hide a pin on the **Parameters** tab in the **Properties** dialog box.



1. In the block, click **Properties** ....

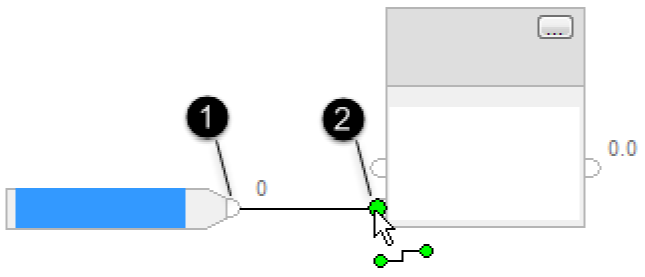


	Vis	Name	Value	Type
I	☐	EnableIn	1	BOOL
I	✓	SourceA	0.0	REAL
I	☐	SourceB		REAL

2. In the **Properties** dialog box, on the **Parameters** tab, clear the **Vis** check box to hide the pin. Select the **Vis** check box to show the pin.
3. Click **OK**.

Wire elements together

Wire (connect) two elements together by clicking the output pin of the first element and clicking the input pin of the other element. A green dot shows a valid connection point.

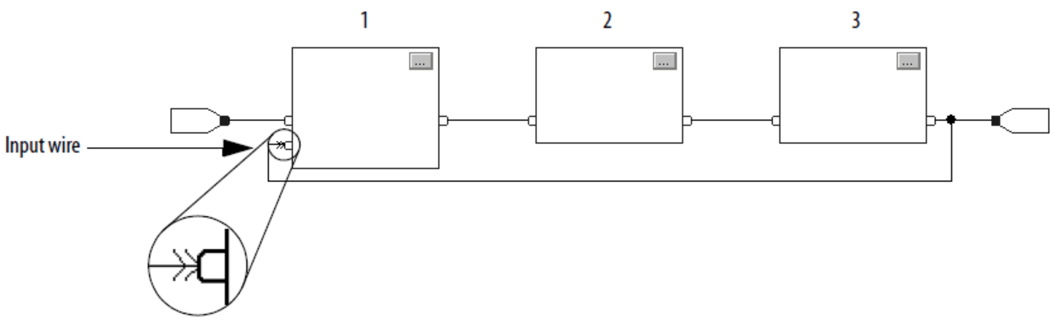


Item	Description
❶	Output pin of the first element
❷	Input pin of the second element

Mark a wire with the Assume Data Available indicator

When there are a group of blocks in a loop, identify which block executes first. The **Assume Data Available** indicator marks the input wire that creates the loop (the feedback wire). It defines the data flow within the loop.

- To define a wire as an input wire, right-click the wire and click **Assume Data Available**.



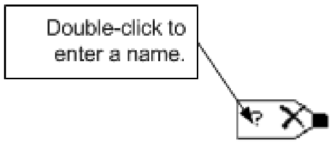
The arrow indicates that the data serves as input to the first block in the loop.

Assign a tag

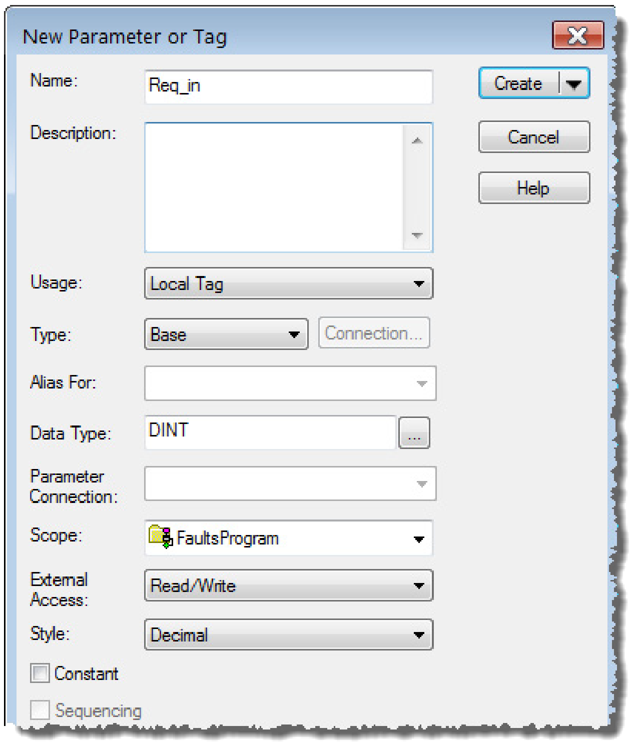
Create and assign a new tag to the connector or assign an existing tag to the connectors.

Create and assign a new tag

1. Double-click the operand area to enter a name.



2. In the box, type a name for the tag and press **Enter**.
3. Right-click the tag name, and click **New <tag name>**.


New Parameter or Tag

Name: Req_in

Description:

Usage: Local Tag

Type: Base

Alias For:

Data Type: DINT

Parameter Connection:

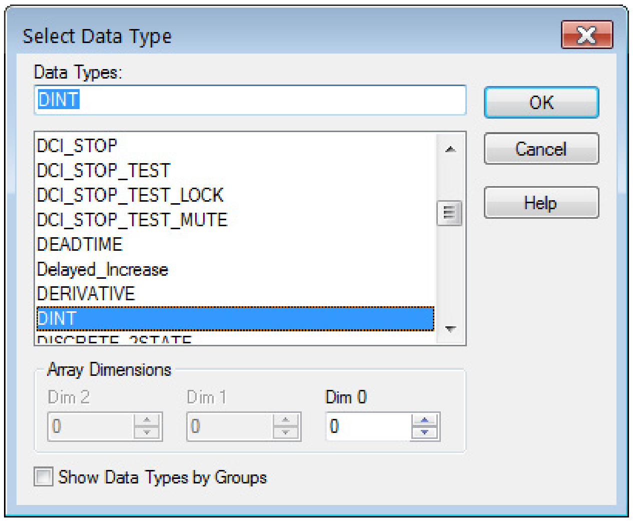
Scope: FaultsProgram

External Access: Read/Write

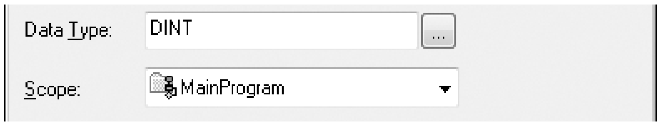
Style: Decimal

4. On the **New Parameter or Tag** dialog box, in the **Usage** list, click a usage value for the tag. The default is a local tag.
5. In the **Type** list, click the tag type.

6. In the **Data Type** box, click ….



7. On the **Select Data Type** dialog box, click the data type for the tag.
8. If the tag is an array, in the **Dim 0** box, type or select the number of elements in each dimension.
9. Click **OK**.
10. On the **New Parameter or Tag** dialog box, in the **Scope** list, click the scope for the tag.

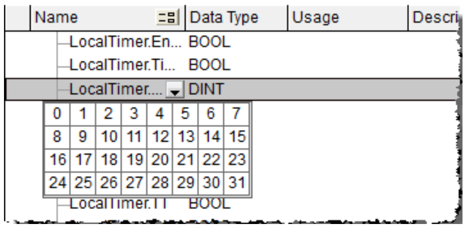


11. Click **Create**.

Assign an existing tag

1. Double-click the operand area.

2. In the box, click the down arrow to select the tag.
3. On the **Tag Browser**, click the tag, or select the bit by clicking the down arrow to the right of the tag and clicking the bit.



4. Press **Enter** or click a different spot on the diagram.

Assign an immediate value (constant)

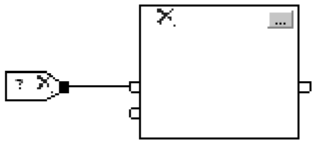
You can assign a constant value instead of a tag value to an input parameter.

If you want to:	Then:
Make the value visible on the diagram and in reports	Use an IREF on page 28
Change the value online without editing the routine	Enter a Value in the Tag of a Block on page 28

Use an IREF

Complete these steps to assign a value to an IREF.

1. Add the IREF to the routine. For instructions on adding an element, see Add a function block element on page 21.
2. Wire the IREF to the input pin that gets the value. For instructions on wiring elements together, see Wire elements together on page 25.

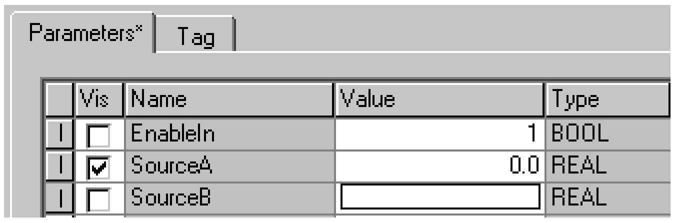


3. Double-click the operand area of the IREF.
4. In the box, type the value and press **Enter**.

Enter a value in the tag of a block

Complete these steps to assign a value to a parameter when on wire connects to its pin.

1. In the block, click **Properties** ....

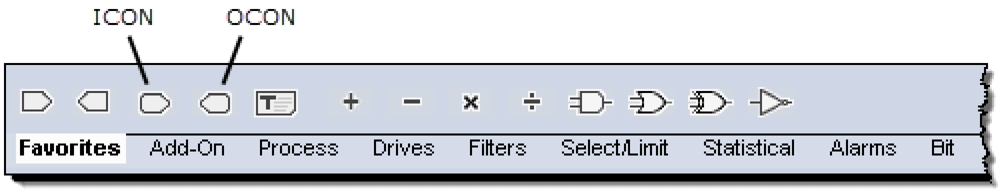


	Vis	Name	Value	Type
I	☐	EnableIn	1	BOOL
I	☑	SourceA	0.0	REAL
I	☐	SourceB		REAL

2. On the **Parameters** tab, in the **Value** box of the desired parameter, type the value.
3. Click **OK**.

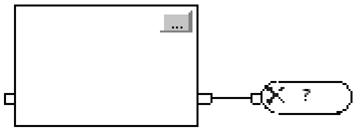
Connect blocks with an OCON and ICON

Use an output wire connector (OCON) or input wire connector (ICON) to transfer data between sheets or in complex wiring situations.



Add an OCON

1. Add an OCON and place it near the output pin that supplies the value.
2. Wire the OCON to the output pin.

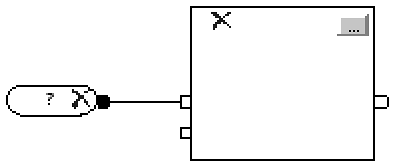


3. Double-click the operand area of the OCON.
4. In the box, type a name that identifies the connector and press **Enter**.

Add an ICON

1. Add an ICON and place it near the input pin that gets the value from the corresponding OCON.

2. Wire the ICON to the input pin.



3. Double-click the operand area of the ICON.
4. In the box, click the down arrow and click the name of the OCON that supplies the value to this connector.
5. Click **Enter** or click a blank spot on the diagram.

Rename a wire connector

Edit the name of an input wire connector or an output wire connector in a routine.

1. Right-click the operand area of the desired ICON or OCON, and click **Rename Element**.
2. In the box, type or select a new name and press **Enter**.

Rename a connector group

Wire connectors that share the same connector name can be changed in a routine.

Tip: If there are wire connectors with the new name, the renamed wire connectors merge with the existing connectors.

To rename a connector group

1. Right-click the operand area of the desired ICON or OCON, and click **Rename Connector Group**.
2. In the box, type or select the new name and press **Enter**.

   All instances in the group are changed to the new name.

Verify the routine

As you program your routine, periodically verify your work.

1. On the **Standard** toolbar, click the Verify icon.

   If there are errors, they are listed in the **Output** window on the **Errors** tab at the bottom of the Logix Designer application.
2. Press **F4** to go to the first error or warning.
3. Correct the error according to the description in the **Errors** tab.
4. Repeat steps 1...3 until all of the errors are corrected.
5. To close the **Output window,** press **Alt + 1.**

Index

Rockwell Automation support

Use these resources to access support information.

Technical Support Center	Find help with how-to videos, FAQs, chat, user forums, and product notification updates.	rok.auto/support
Knowledgebase	Access Knowledgebase articles.	rok.auto/knowledgebase
Local Technical Support Phone Numbers	Locate the telephone number for your country.	rok.auto/phonesupport
Literature Library	Find installation instructions, manuals, brochures, and technical data publications.	rok.auto/literature
Product Compatibility and Download Center (PCDC)	Get help determining how products interact, check features and capabilities, and find associated firmware.	rok.auto/pcdc

Documentation feedback

Your comments help us serve your documentation needs better. If you have any suggestions on how to improve our content, complete the form at rok.auto/docfeedback.

Waste Electrical and Electronic Equipment (WEEE)

At the end of life, this equipment should be collected separately from any unsorted municipal waste.

Rockwell Automation maintains current product environmental information on its website at rok.auto/pec.

Rockwell Automation Publication 1756-PM009I-EN-P - September 2020
Supercedes Publication 1756-PM009H-EN-P - November 2018