L13 - Studio 5000® and Logix Advanced Lab

For Classroom Use Only!
Important User Information

This documentation, whether, illustrative, printed, “online” or electronic (hereinafter “Documentation”) is intended for use only as a learning aid when using Rockwell Automation approved demonstration hardware, software and firmware. The Documentation should only be used as a learning tool by qualified professionals.

The variety of uses for the hardware, software and firmware (hereinafter “Products”) described in this Documentation, mandates that those responsible for the application and use of those Products must satisfy themselves that all necessary steps have been taken to ensure that each application and actual use meets all performance and safety requirements, including any applicable laws, regulations, codes and standards in addition to any applicable technical documents.

In no event will Rockwell Automation, Inc., or any of its affiliate or subsidiary companies (hereinafter “Rockwell Automation”) be responsible or liable for any indirect or consequential damages resulting from the use or application of the Products described in this Documentation. Rockwell Automation does not assume responsibility or liability for damages of any kind based on the alleged use of, or reliance on, this Documentation.

No patent liability is assumed by Rockwell Automation with respect to use of information, circuits, equipment, or software described in the Documentation.

Except as specifically agreed in writing as part of a maintenance or support contract, equipment users are responsible for:

• properly using, calibrating, operating, monitoring and maintaining all Products consistent with all Rockwell Automation or third-party provided instructions, warnings, recommendations and documentation;
• ensuring that only properly trained personnel use, operate and maintain the Products at all times;
• staying informed of all Product updates and alerts and implementing all updates and fixes; and
• all other factors affecting the Products that are outside of the direct control of Rockwell Automation.

Reproduction of the contents of the Documentation, in whole or in part, without written permission of Rockwell Automation is prohibited.

Throughout this manual we use the following 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.

**IMPORTANT**

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

**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
• recognize the consequence

**SHOCK HAZARD**

Labels may be located on or inside the drive to alert people that dangerous voltage may be present.

**BURN HAZARD**

Labels may be located on or inside the drive to alert people that surfaces may be dangerous temperatures.


Contents

Before you begin ........................................................................................................................................... 5
About this lab .................................................................................................................................................. 5
Tools & prerequisites ..................................................................................................................................... 5

Lab 1: Usability and Work Flow Enhancements <35 minutes> ................................................................. 7
   1.1 Logix Designer New Look and Feel .................................................................................................. 7
   1.2 New Structured Text Editor ................................................................................................................ 11
   1.2 New Function Bock Diagram (FBD) Features ................................................................................. 28
   1.3 Detect a Lonely Routine with Verify ............................................................................................... 36
   1.4 New On-Line Power Rail Display .................................................................................................... 38
   1.5 Output Window ................................................................................................................................ 45

Lab 2: Introduction to the Logical Organizer <10 minutes> ...................................................................... 49
   2.1 Using the Logical Organizer ................................................................................................................ 49
   2.2 Right Click Find in Organizer ........................................................................................................... 53

Lab 3: Compare and Merge Tool <15 minutes> ....................................................................................... 56
   3.1 Open the Logix Designer Compare Tool .......................................................................................... 57
   3.2 Comparing Project Files .................................................................................................................... 59
   3.3 Comparing Project Exports ................................................................................................................. 63
   3.4 Merging Project Files ......................................................................................................................... 67
   3.5 Saving Merged Project ........................................................................................................................ 73
   3.6 Reviewing Merged Project .................................................................................................................. 75

Lab 4: Using AOI’s to Monitor Module Status <15 minutes> .................................................................... 76
   4.1: Using Module as an InOut Parameter ............................................................................................ 76
   4.2: Using the Module Object Path Attribute ....................................................................................... 83

Lab 5: Improving Efficiency Using Partial Import Online <30 minutes> .................................................. 89
   5.1: Code Library Example .................................................................................................................... 89
   5.2: Line Expansion Example .................................................................................................................. 97
5.3: PIO Search and Replace Example ................................................................................................................................. 112

Lab 6: Introduction to Program Parameters <40 minutes> ........................................................................................................ 127
6.1: Input Parameters ................................................................................................................................................................. 128
6.2: Output Parameters ............................................................................................................................................................... 136
6.3: InOut Parameters ................................................................................................................................................................. 143
6.4: Public Parameters ................................................................................................................................................................. 146
6.5: Direct Access of Program Parameters ................................................................................................................................ 151

Lab 7: Add-On Instructions (AOI) <25 minutes> ........................................................................................................................ 155
7.1 AOI Basics ............................................................................................................................................................................. 156
7.2 Creating an Add-On Instruction (AOI) ................................................................................................................................. 160
7.3 Entering the Logic ................................................................................................................................................................. 165
7.4 Using the AOI in RLL (Ladder Logic) ................................................................................................................................ 167
7.5 Using the AOI ........................................................................................................................................................................ 169
7.6 Creating Alias Tags ............................................................................................................................................................... 172
7.7 Using an Add-On Instruction Output in Ladder ................................................................................................................... 174
7.8 Using the AOI in a Function Block (FB) Routine ................................................................................................................... 175
7.9 Re-Using an AOI ................................................................................................................................................................. 178
7.10 AOI Signatures .................................................................................................................................................................... 182
7.11 Export AOI ........................................................................................................................................................................ 186

Lab 8: Using a SD card with a Logix Controller <10 minutes> ............................................................................................... 192
8.1 Opening an Existing Controller Project ............................................................................................................................ 192
8.2 Using code to list files and directories ................................................................................................................................. 194
8.2 Create a Directory .............................................................................................................................................................. 198
8.3 Create, Edit, and Delete a File ............................................................................................................................................. 199
8.4 Read Data from SD card ....................................................................................................................................................... 201
8.5 Storing and Loading Recipe Data ....................................................................................................................................... 203

Lab 9: Controller Change Log <10 minutes> ............................................................................................................................ 207
Before you begin

About this lab
Welcome to this Hands-On Lab series! This session provides you with an opportunity to explore advanced concepts within the Logix platform. The following sections explain what you’ll be doing in this lab session, and what you will need to do to complete the hands-on exercises.

You will have approximately 90 minutes to complete as many sections of this lab as you would like too. It will take approximately 180 minutes (3 hours) to complete all of the sections! You do not need to complete all of the sections.

The intent of this lab to present an array of topics for the user to select from; each lab section is independent so you can pick and choose what you are interested in during the time available. If you choose to complete every lab section it will take more than the allotted time for the lab.

NOTE: To ensure proper lab execution of the lab the VMWare image should be reset to the correct snapshot prior beginning the lab.

Tools & prerequisites
This hands-on lab is intended for individuals who use:

Studio 5000® Logix Designer or RSLogix 5000
ControlLogix or CompactLogix

This hands-on lab uses the following files:

Lab 1 Usability and Workflow Enhancements -> Usability_Enhancements.ACD
Lab 2 Introduction to the Logical Organizer -> Logical_Organizer.ACD
Lab 3 Compare and Merge Tool -> Line_Expansion.ACD
Lab 4 Improving Efficiency Using Partial Import Online -> Tank_Controller.ACD
Lab 5 AOI Hardware Monitoring -> Tank_Supervisor.ACD
Lab 6: Introduction to Program Parameters -> Station_Section3.L5X
Lab 7 Add-On Instructions AOI -> Hardware_Monitoring_ModuleSts.ACD
Lab 8 Using a SD card with a Logix Controller -> Hardware_Monitoring_ModuleWho.ACD
Lab 9 Controller Change Log -> Sample_Project_Start.ACD
Files to be placed on SD card

Lab 8 Using a SD card with a Logix Controller
-> CLX_S7_00.ACD
-> CLX_S8_00.ACD
-> CLX_S13_00.ACD
Files to be placed on SD card

Lab 9 Controller Change Log
-> CLX_S14_00.ACD
-> ControllerLogSample.txt
This hands-on lab uses the following hardware:

ControlLogix Demo Case (1756_L85E, 1756_OB16IEF, 1756_IB16IF, 1756_IF8I, 1756_OF8I)

This hands-on lab uses the following Software:

Studio 5000® Logix Designer v31.00.00

Studio 5000® Logix Designer Compare Merge Tool
Lab 1: Usability and Work Flow Enhancements <35 minutes>

This section will provide an introduction to some of the features that are to be introduced in version 31 of Studio 5000® Logix Designer. These changes work together to give you a large increase in usability. All of these changes came from customer input.

Goals:
- Logix Designer New Look and Feel
- New Structured Text Editor!
- Function Block Diagram (FBD) updates

Updates from v28:
- Delete a program without having to unschedule it
- New on-line power rail display

1.1 Logix Designer New Look and Feel

1. Double-click on the Lab Files shortcut on the desktop.

2. Navigate to C:\Lab Files\Advanced Logix\Lab 1 Usability Enhancements.

3. Open the project named Usability_Enhancements.ACD.

NOTE: This same ACD file will be used for each section of this lab. So, keep it open until you are instructed to close Studio 5000® Logix Designer.
Many items have changed, but many items have stayed the same!

In v31 colors and fonts have changed, but the names have stayed the same.

**Top Menu**
Menu items are identical to v30. Display font has changed, but item names are all the same.

**Controller Organizer**
Naming and Organization is similar as v30. Icons and fonts have changed.

**Logical Organizer**
Naming and Organization is the same as v30. Icons and fonts have changed.

**Toolbars**
Icons have been updated, but selections are the same.

4. Explore the New Look and Feel!
Colors, Icons, instruction displays have all been updated.

5. Open any Routine from the Controller Organizer.

6. Routine Tabs are now shown at the top.
7. You can now drag a routine out of the Logix Designer framework. Select the CommSts_Local – MainRoutine Tab with your mouse and drag it over to the right.

8. Now the Routine that you opened is shown in a separate window.

9. In the example above CommSts_Local – MainRoutine Tab has been moved into a separate window.

10. Open at least 5 additional routines (do more if you like).

11. Click the down list button on the far right of the Routines tabs. This will then bring up a list of all of the Routines that are currently open.
12. Hover the cursor over the Find Next Icon in the toolbar.

Information provided in the Tool Tips has been expanded.

**Tool Tips Now Have Expanded Descriptions**

New in v31 tool tips have been expanded to include:

- Icon of the selection
- Selection name
- Shortcut key for selection
- Description of selection

*Expanded information is not just for toolbar icons. Tool Tips descriptions have been expanded throughout Logix Designer!*

1.2 New Structured Text Editor

In this sub section we will explore the enhancements made in the completely new Structured Text Editor.

- Line Numbers
- Change and Verify Bars
- Code Outlining
- Completion Prompt
- Smart Indent
- Code Snippets
- Inline Value Monitoring

Other New Structured Text enhancements:

- Header Auto Populate
- String to String Literal Assignment
- String to String Literal Comparison

**Line Numbers**

13. Expand the Structured_Text Program folder.

14. Open the routine STX_Routine.

```
1 /////////////////////////////////////////////////////////////
2 // Copyright (c)
3 //----------------------------------------------
4 // Routine: STX_Header
5 // Author: <PC_Name>/<User_Name>
6 // Created: 4/26/2017
7 //----------------------------------------------
8 // Description:
9 // Structured Text Header description text is the
10 // Routine description.
11 //
12 //
13 // History:
14 // Created for Rockwell Automation Advanced Logix Lab
15 //--------------------------------------------------------
```

Line Numbers are now displayed on the left hand side of the routine display.
15. Open the Controller Tags

16. Select the tag Product_Selection

17. Right Click and Select Go to Cross Reference for “Product_Selection”

18. Tag Cross References now include the Structured Text Line Number as well as the Ladder Rung Number.
Structured Text Editor Display Configuration

You access Workstation Options from the top menu Tool → Options...

Many display items can be configured. Line numbers can be turned off, etc...
Change and Verify Bars

19. Open the STX_Routine in the Structured_Text Program.

20. Select Line 16, this is an empty line.

21. Press the Enter key twice

You will now see two yellow bars on the lines that you have added. Lines 16 through 18. The yellow bars indicate that changes have been made and that they have not been verified.
22. Verify the STX_Routine

23. Now the change bar color is green indicating the changes have been verified.

Tracked Changes color bars are removed when the Routine is closed.
**Code Outlining**

24. Go to Line 20 in STX_Routine.

25. Click the – box to the Left of “if” to contract the code region.

26. Hover the cursor over the ellipse box.

27. This will bring up a Tool Tip that contains all of the text that is hidden by the collapsed region.

28. The use of the #Region and #Endregion syntax indicates a User Defined Region of code.

29. Go to Line 38 in STX_Routine.

30. Use the – box to the Left of #Region to contract the code region.

31. The Region syntax is now hidden and the user only sees the + box and the description that was entered to the right of #Region.
32. Like the if...then example above, you can bring up a tool tip that contains all of the text in the hidden region. Do this by hovering the cursor over the box containing the hidden region.

33. You can also create a very large User Defined Region of code.

34. Go to Line 35 in STX_Routine.

35. Close the Region

36. This is an example of a very large region that closes lines 37-68. Since this region is very large the tool tip is not able to display all of the text that is hidden.
**Code Snippets**

37. Open the Routine STX_Code_Snippet

**What are Code Snippets?**

A code snippet inserts a predefined syntax with template parameters.

To insert a code snippet:

1. Type in first word of the syntax.
2. Press the Tab key
3. Code snippet is automatically inserted.

Supported Constructs:

- IF… THEN
- ELSEIF… THEN
- CASE… OF
- FOR… DO
- WHILE… DO
- REPEAT… UNTIL

When a Code Snippet is inserted placeholders tags are added. Placeholder tags are highlighted in yellow and must be replaced with the tags you will use for your Routine.

Tool Tips can be used to further explain what the placeholder tag is used for.
38. Go to Line 17, this is the second blank line after the header.

39. Type “if”

40. Press the TAB key

41. An ‘if..then’ code snippet is now added.

```
17 if bool_expression then
18 |
19 end_if;
```

42. Replace bool_expression by typing ‘Product_Running’. You do not need to move the cursor before typing.

```
17 if Product_Running then
18 |
19 end_if;
```

43. Enter the expression in Line 18: System_Status := ‘Running’;

```
17 if Product_Running then
18     System_Status := 'Running';
19 end_if;
```

44. Your if..then Structured Text code is now complete.

45. Experiment with code snippets for the the other syntaxes. Type the first word or first few characters of the syntax and then press Tab.

   a) ELSEIF... THEN
   b) CASE... OF
   c) FOR... DO
   d) WHILE... DO
   e) REPEAT... UNTIL
Automatically Insert a Header with Auto Populated Fields

46. Click on Tools in the top menu and select Options….

47. The Workstation Options dialog window is shown.

48. Under the Categories list, select Structured Text Editor.
49. The **Change Structured Text Editor Preferences** dialog allows you to configure how the Structured Text Editor works.

### Structured Text Editor Preferences

<table>
<thead>
<tr>
<th>Preference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Auto-indent</strong></td>
<td>Automatically sets the indentation of new lines at the same value</td>
</tr>
<tr>
<td><strong>Tab Size</strong></td>
<td>The number of spaces that each tab represents. Press the Tab key to insert a tab</td>
</tr>
<tr>
<td><strong>Insert Paces</strong></td>
<td>Creates spaces instead of tabs (ASCII spaces)</td>
</tr>
<tr>
<td><strong>Keep Tabs</strong></td>
<td>Inserts regular tabs. (ASCII tab character)</td>
</tr>
<tr>
<td><strong>Enable Duplicate Destructive Bit Detection</strong></td>
<td>Check box if you want to receive a warning whenever a bit referenced by an OTE, ONS, OSF, or OSR instruction is destructively referenced elsewhere in the controller.</td>
</tr>
<tr>
<td><strong>Include Header on Create</strong></td>
<td>Enables text header to be automatically added to the beginning of new Structured Text routines. You can edit the text that is automatically inserted in the text box below this checkbox.</td>
</tr>
</tbody>
</table>

50. Check the **Include Header on Create** checkbox. This will automatically create a header in any newly created Structured Text Routine.

51. Edit the information inside the automatic header. Information in <> is automatically populated.
52. Click **OK**.

53. Create a new Structured Text Routine named **STX_Header** inside the **Structured_Text** Program.

You can see that Logix Designer has automatically created a header!

NOTE: The Routine type must be set to Structured Text.

54. Open the new routine. You can see that Logix Designer has automatically created a header!
55. Open the **No_Header** Routine inside the **Structured_Text_Header** Program. This routine was created before the **Include Header on Create** checkbox was checked.

**NOTE:** This is the default setting.
String to String Literal Assignment

56. Open the **STX_Routine**

57. Go to Line 38 and expand the region if needed.

58. Lines 39 through 42 give us an example of a String literal to String assignment.

```plaintext
39 Str_Product_List[0] := 'Chocolate Chip';
40 Str_Product_List[1] := 'Mint Chocolate';
```

**String Literal Assignment**

String Literal Assignment allows the user to set tags of the string data type to a hard coded value. We have been able to do this to Booleans, DINT, REAL, etc. data types. Now this functionality is available for STRING data type tags.

You can also assign String Literals to a STRING data type tag in Ladder. This can be done with a MOV instruction.

**NOTE:** String Literal Assignment and Comparison features are only applicable to:

- ControlLogix 5580
- CompactLogix 5380 & 5480
String to String Literal Comparison

59. In the **STX_Routine**, go to Line 46.

60. The syntax `Str_Product = 'Chocolate Chip'` is an example of a String literal to String Comparison.

```
46  if Str_Product = 'Chocolate Chip' THEN
```

---

**String Literal Comparison**

String Literal Comparison allows the user to compare a tags of the string data type to a hard coded string value. We have been able to do this to Booleans, DINT, REAL, etc. data types. Now this functionality is available for STRING data type tags.

This is an ASCII character value comparison.

You can also assign String Literals to a STRING data type tag in Ladder. This can be done with an **EQU** instruction.

---

**Putting it All Together**

Depending on your work from the earlier section, the pictured line numbers may differ.

61. Open the **STX_Code_Snippets** Routine in the Structured_Text Program.

62. Go to the last empty line of the Routine.

63. Press Enter to insert a blank line

64. Type ‘if’ then press the TAB key. This will give you the if…then Code Snippet.

```
16  if bool_expression then
17     end_if;
```
65. We will then replace the 'bool_expression' placeholder text with the following expression:
   a)  \textbf{Str\_Product} = 'Sugar Cookie'
       
       17 if Str\_Product='Sugar\_Cookie' then
       19 end if;
   b) This is an example of a String Literals Comparison.

66. In the blank line, enter the follow expression:
   a) \textbf{Str\_Product\_List[4]} := 'Sugar Cookie';

   b) This is an example of a String Literals Assignment.

67. Verify the STX\_Code\_Snippets Routine.
   a) Address any errors.

68. Go to line 16 of the Routine.

69. Press Enter to insert a blank line

70. Type \#Region Set Product to Sugar Cookie

71. Go to the last line of the Routine.

72. Press Enter to insert a blank line

73. Type \#EndRegion

74. Press Enter

75. You have now added a user defined code region in the Routine. Close the region.
**Inline Value Display**

Tag values are now displayed under the tag. These values are shown with a grey highlight. Values are shown both offline and online.

You can also hover over tags to see

Note: You can turn Inline Value Display Off in Workstation Options.

You can also hover over tags to see their value!

76. Open the **STX_Routine**

77. Select the **Toggle Inline Value Display** option.

78. The inline values are now displayed below the tags.

79. Close Studio 5000. If prompted to save the file respond with **No.**
1.2 New Function Bock Diagram (FBD) Features

In this sub section we will explore the enhancements made in the Function Bock Diagram (FBD) editor.

- Modify operand values from FBD sheet

Other New Structured Text enhancements:

- Organize FBD sheets while Online

### Coming Soon – FBD Modify Sheet Size while Online

V31 will allow you size FBD Sheets while online!

Also, v31 changes the default FBD sheet size to 11x17 Landscape.

*Note: Pre-Release information is subject to change.*

**Modify Operand Values from FBD sheet**

80. Double-click on the [Lab Files](#) shortcut on the desktop.

81. Navigate to C:\Lab Files\ Advanced Logix\Lab 1 Usability Enhancements.

82. Open the project named [Usability_Enhancements.ACD](#).

83. Download the Usability_Enhancements.ACD project to the 1756-L85E demo case controller.

84. Click on Communications => [Who Active](#) at the top of the screen.

*NOTE: RSLinx may take a few moments to open.*
85. Expand the Ethernet/IP Driver AB_ETHIP-1.

86. Select 192.168.1.218, 1756-L85E. You do not need to expand the controller.

87. Click Download.

88. When prompted, click Download.

89. If prompted to change the controller back to Remote Run, Click Yes.

90. If not prompted, change the controller mode to Remote Run in Logix Designer.

91. Expand the _1100_Boiler Program.

92. Open the AT1131 Routine
93. Double Click the value of Input Reference Rack_01_01_A1.Ch0Data

94. Change the value to 80.8

95. Press Enter

96. You can open other FBD Routines and experiment with changing operand values while online.
Organize FBD Sheets while Online

97. Expand the program Sim_1100_Boiler

98. Double-click on the FBD routine Boiler1Sim. This will open FBD routine on Sheet 1 named Fuel.

99. Click on the Start Pending Routine Edits icon in the FBD toolbar.
100. Click on the **Organize Sheets** icon in the FBD toolbar

![Organize Sheets icon in FBD toolbar](image)

**Accept and Cancel Edits buttons are active indicating that we can edit the FBD.**

101. From the **Organize Sheets** dialog box we can select a sheet name and use the mouse to move sheets up or down to re-order them.

![Organize Sheets dialog box](image)

**Fuel sheet name is selected. Use the mouse to drag it down the list.**

102. While moving the sheet a green line will indicate where the sheet will be placed.

![Organize Sheets dialog box](image)

**Fuel sheet name is selected. Use the mouse to drag it down the list.**
103. In the screen capture below, the sheet named Fuel was moved from 1 to 3. Note that the sheet number did not change. It will change to 3 after the **OK** button is clicked.

![Organize Sheets dialog box](image)

Sheet number still at 1. It will change to 3 once the **OK** button is pressed.

104. Click the **OK** button and verify that the sheets have been re-ordered.

105. Click **Accept Pending Routine Edits** button

106. The sheets have now been re-ordered! You can use the **Organize Sheets** dialog box to move the order of other sheets in this Project.

**NOTE:** *Keep the Logix Designer project open.*
Delete a Program Without Having to Unschedule It

107. Go Offline with Logix Designer

108. Select the **CommSts_Local** Program.

109. Right click on the **CommSts_Local** program and select **Delete**.
    
NOTE: The program CommSts_Local is currently scheduled to run as the first program in the MainTask.

110. Click **Yes** on the dialog to delete the program.

111. That's it! Program is completely deleted.
Removing Programs from a Logix Designer project has gotten much easier!

Prior to Logix Designer v28, you needed to follow a long and tedious workflow to delete a program.

Program deletion steps prior to v28:
1. Unschedule the Program from the Task
2. Individually delete all of the Routines
3. Manually delete all of the Program Scope Tags
4. Right-Click on the Program and select delete

Program Deletion Step for v28 and greater:
1. Right-Click on the Program and select delete

As you can see, the updated workflow is much simpler and more intuitive!
1.3 Detect a Lonely Routine with Verify

112. Select the Program **Lonely_Routine**.

113. Right-click and select **Verify**.

114. View the **Errors** window at the bottom of Logix Designer (use the scroll bar to scroll up to see warnings). Note the Lonely Routines warnings that were found.

---

**What are Lonely Routines?**

Lonely Routines are routines that are not called by their program’s Main Routine. The Lonely Routine Verify Warning alerts you to a routine that will not be executed by the Logix controller. This would be the same as being unscheduled.

A Logix controller's application code scan starts with Tasks. Tasks are scheduled or continuous.

Task contain Programs. The order Programs are scanned in a task is set by the user.

Each Program has a Routine configured as Main. The Main routine is the only routine that the Logix controller automatically scans. All other Routines in a Program need to be called by the user application code.

So, a Lonely Routine warning alerts you to a routine that is not being called automatically by Logix or the user application code!
115. Check the logic to see if these are Lonely Routines by opening the **Main_Routine** of the **Lonley_Routine** Program. Notice there are no call instructions in this routine or the other Lonely Routines.

   NOTE: A call to a Routine would be one of the following instructions: JSR or FOR.

116. You can also **Cross Reference** each routine to search for calls.

117. In the results of the Cross Reference of **the Lonely_Ladder_Routine** you can see that nothing was found. This indicates that no instructions reference the Lonely_Ladder_Routine. Since no instructions reference the Lonely_Ladder_Routine, no calls are being made to execute logic in the Lonely_Ladder_Routine.
1.4 New On-Line Power Rail Display

118. Click on **Communications** => **Who Active** at the top of the screen.
   
   NOTE: RSLinx may take a few moments to open.

119. Expand the Ethernet/IP Driver **AB_ETHIP-1**.

120. Select **192.168.1.218, 1756-L85E**. You do not need to expand the controller.

121. Click **Download**

122. When prompted, Click **Download**.

123. If prompted to change the controller back to **Remote Run**, Click **Yes**.

124. If not prompted, change the controller mode to **Remote Run** in Logix Designer.

125. Expand the **MainTask** and then expand the Program **_1100_Boiler**

126. Open the Routine **AIC1131_O2_LAD**
From this online view of the Logix Designer ladder editor you will see green power rail on the right and left hand sides of the ladder rungs.

NOTE: This is the same green Power Rail display that has been used in all Logix versions.
127. Open the routine **Lonely_Ladder_Routine**

128. Notice how this routine does not have a green power rail. As we saw in the Lonely Routine section of this lab, the **Lonely_Ladder_Routine** is not being scanned by the Logix controller.

   NOTE: The screen capture above was taken online. Since the routine is not being scanned, its power rails are the same as an offline application.

129. Expand the program **_1100_Boiler**

130. Open the routine **MainRoutine**

131. Delete **Rung 0**

132. Right click on rung 0 and choose **Accept Pending Program Edits**.

133. Next, select the **Test Accepted Program Edits** button on the routine’s toolbar.
134. Click Yes to the following dialog to test the program edits.

![Dialog Image]

135. Your rung should look like the one below.

NOTE: This change needs to be made online.

![Rung Image]

136. Now the routine **AIC1131_O2_LAD** is not being called by any programs in our online project.
137. Open the routine **AIC1131_O2_LAD**

138. Notice how this routine does not have a green power rail. As we saw in the *Lonely Routine* section of this lab, **AIC1131_02_LAD** is not being scanned by the Logix controller.

   **NOTE:** The screen capture above was taken online. Since the routine is not being scanned its power rails are the same as an offline application.

139. Click the **Untest Accepted Program Edits** button on the routine toolbar. Click **Yes** to the dialog that appears.

140. Click **Cancel Accepted Program Edits**. Click **Yes** to the dialog that appears.
141. Open the Routine **MainRoutine** in Program **_1100_Boiler**

142. In Rung 0, insert an AFI instruction before the JSR instructions.

143. Using the buttons on the routine toolbar, **Accept** and **Test** the rung edit. Your rung should look like the one below.

   **NOTE:** This change needs to be made online.

144. Now the routine **AIC1131_O2_LAD** is not being called by any programs in our online project.

145. Open the routine **AIC1131_O2_LAD**
146. Notice how this Routine still has a green power rail. Why is this?

**Limitations of the Online Power Rail Display**

Studio 5000® Logix Designer can detect when there are no calls to a Routine. But, conditional calls to a Routine will always contain Green Power Rails.

The same scenario would be true if we used a timer to periodically call a Routine.

147. Click the **Untest Accepted Program Edits** button on the routine toolbar. Click **Yes** to the dialog that appears.

148. Click **Cancel Accepted Program Edits**. Click Yes to the dialog that appears.
1.5 Output Window

149. Select the Program **Lonely_Routine** in the **Controller Organizer**.

150. Right-click and select **Verify**.

151. Make note of the new filter buttons

152. Click on the **6 Messages** box.
   This will toggle off the updates that are status messages.

153. Select the **6 Messages** box and de-select the **4 Warnings** box.
   This will enable the status messages and toggle off the warnings.

154. De-select the **6 Messages** box.
   This will filter to only the Errors but since there are not errors none are displayed.

155. Navigate to the **Lonely_Ladder_Routine** in the **Lonely_Routine** program.
156. Insert a blank rung into the routine.

157. Close the **Lonely_Ladder_Routine**.

158. Select the Program **Lonely_Routine** in the **Controller Organizer**, right-click and select **Verify**.

159. Configure the error window to display the Errors and Warnings.

160. With the Error window active hit F4 to cycle through the list.
    Shift + F4 moves backwards through the list.

    As you move through the warnings on the list, the noted routine is highlighted.
161. Navigate to the entry **Error: Rung 2: Empty Rung.**

When you got to the error on the list the routine that contained the error was automatically opened and the rung selected.

**Updated Output Window**

The Output window appears at the bottom of the Logix Designer application main window when an operation yields multiple results or errors. The Output window also appears when a routine is opened online. The Output window contains Errors, Search Results, and Watch tabs.

The Output Window added filtering functionality in version 31. The filtering allows you to display a combination of errors, warnings or messages. As noted in the lab, you can toggle the corresponding button to enable/disable the displayed items. For example, to display the errors and messages, click the Errors and Messages buttons.
162. Close the Project.

Congratulations!
You have just finished this Lab
Click to Proceed to Contents
**Lab 2: Introduction to the Logical Organizer <10 minutes>**

This section will provide an introduction to a feature that was introduced in version 24 of Studio 5000® Logix Designer. You will be organizing the project from Introduction to Program Parameters section into logical groups.

**Goals:**
- Organize the Introduction to Program Parameters section project.
- Understand the different options available within the Logical Organizer.

### 2.1 Using the Logical Organizer

1. Double-Click on the **Lab Files** folder on the desktop.

2. Navigate to **C:\Lab Files\Advanced Logix\Lab 2 Introduction to the Logical Organizer**

3. Open the project named **Logical_Organizer.ACD**.
   
   **NOTE:** *This may take up to a minute to open.*
4. Locate and select the **Logical Organizer** tab in the bottom left window.

**NOTE:** The Controller Organizer is used to organize programs into a task structure that defines how the logic will be executed. This view doesn't always present code modules in a logical format that is applicable to the machine or process design. The Logical Organizer gives users a new way to group and view programs in an application. This can be extremely helpful when viewing code modules that have been developed at the program level. The remainder of this section will walk you through the Logical Organizer.
5. Click on the **Logical Organizer** tab.

6. Right-click on the root folder *(Logical Model Tank_Controller)* and Select **Add => New Folder…**

7. Name the folder **Simulation_Logic**.
   
   NOTE: This folder will hold the simulation program used to fill and drain the tank.

8. Click **OK**.

9. Create two additional folders using the same steps. Name one folder **Tank_Manager** and the other **Station_1**.

Now that we have some folders created, let’s move the programs around.
10. Click on the **Simulation1** program and drag it to the **Simulation_Logic** folder.

11. Click on the **Station_1** folder and drag it to the **Tank_Manager** folder.

12. Click on the **Auto_Cycle1** Program and drag it to the **Station_1** Folder.

13. Click on the **Pump1** Program and drag it to the **Station_1** Folder.
14. Click on the **Tank1** Program and drag it to the **Station_1** Folder.

![Logical Organizer](image)

**General Information about the Logical Organizer**

The Logical Organizer allows the user to separate the application into a multi-level, logical hierarchy of folders, programs, and phases. This view also supports the same functionality as the Controller Organizer to view, create, or delete code. The Logical Organizer also allows you to drag / drop programs and program structures (sub-programs) within the same project or across multiple instances of Studio 5000® Logix Designer. This is particularly powerful if a code library exists in the form of an ACD file. You will see an example of this functionality later in the lab.

**Logical Organizer Facts:**

- New programs that are created in the Logical Organizer will appear in the Unscheduled Programs / Phases folder in the Controller Organizer or can be assigned to a Task.
- The Logical Organizer hierarchy is stored in the controller upon a download.
- Up to 15 sub-programs can be nested under one program or folder.
- The number of programs that can be added to a task has been increased from 100 to 1000.

15. Change to the **Controller Organizer** and verify the results!

   **NOTE:** There are no changes to the Controller Organizer. Changes made in the Logical Organizer do not change the organization in the Controller Organizer.

2.2 **Right Click Find in Organizer**

You can quickly find a Task, Program or Routine in the Controller Organizer by right clicking on the Task, Program or Routine in the Logical Organizer. The same is true in reverse!

16. Open the Logical Organizer and select the **Tank1** program.

   **NOTE:** You will need to navigate to Tank_Manager ➔ Station_1 ➔ Tank1
17. Right-click on **Tank1** and select **Find in Controller Organizer**.

Logix Designer will take you to the **Tank1** program in the Controller Organizer.
18. Right-click on **Tank1** and select **Find in Logical Organizer**

Logix Designer will then take you to the **Tank1** program in the Controller Organizer.

19. Close the project.

**Congratulations!**
You have just finished this Lab

[Click to Proceed to Contents]
Lab 3: Compare and Merge Tool <15 minutes>

Overview of the Logix Designer Compare Tool

Long gone are the days when a single controls engineer completed a project all by themselves. With today’s complex control systems it’s not uncommon to have 2-5 engineers working on their specific parts of the project and chances are they are working from different Logix Designer projects. Wouldn’t it be nice if you had a tool that could compare and merge these different projects together and lets you control the granularity to choose what parts of the project to merge?

Since V17, the Compare Tool has been available to help you find differences in Logix projects. This tool is also available in Studio 5000® with enhancements to accommodate the new features in V31. In this lab we will cover the new Compare and Merge capabilities and then explore the ability to create a new merged file from the result of the compare.

At several points during this lab you will be prompted with:

Unless it is specifically stated, you do not need to save the projects.
3.1 Open the Logix Designer Compare Tool

1. From the Start menu, select **All Programs > Rockwell Software > Logix Designer Tools > Logix Designer Compare Tool**

   The Compare Tool window will appear:

   ![Compare Tool Window](image)

2. In the toolbar Click, the **New Compare** icon.

   ![New Compare Icon](image)

   The **New Compare** pop-up will appear.
This window allows us to pick the two projects we want to compare.
NOTE: The project files are located at C:\ Lab Files\Logix Advanced Lab\

3. For the Left Content: click the Browse button, navigate to the Lab3 Compare and Merge Tool folder and select the L75_Project_Start.ACD file using the path above.

4. For the Right Content: follow the same path and select the Line_Expansion.ACD file.

NOTE: Notice the options provided. We will stay with the defaults here but you could eliminate tag values and include descriptions as part of the compare. You could also filter out some of the tags.

5. Click OK. The Compare Progress window will appear while the compare is occurring.
3.2 Comparing Project Files

6. When the compare process is done, the Compare Results window will show where the differences were found. You may need to expand the window to make it large enough to see everything.

The left most column, titled **Compare Summary**, shows the categories where differences were found. The middle and right columns report how many differences were found in each category.

---

**Color Codes Used in the Information Display**

Information displayed in the Compare Summary, Left Project, and Right Project is color-coded:

- **Red**  
  Indicates items that exist in one project but not the other.

- **Blue**  
  Indicates items that are common between the two projects, but have a difference. For example, you might have a ladder routine that exists in both projects, but with a rung that does not match, or a tag that exists, but contains different data. Double-clicking on these items brings up more detailed compare information.

- **Black**  
  Indicates common items that contain child elements.

- **Gray**  
  Indicates common items in which no differences were found. These are for context only; no further details can be displayed. Logix Designer can detect when there are no calls to a Routine. Conditional calls to a Routine will always contain Green Power Rails.
7. Since the **Properties** entry under the **Controller** folder is **Blue**, some differences exist between the properties of the controllers.

8. Double-click on **Properties**.

9. The difference is also highlighted in blue. In this case the only difference is the Controller Properties **Name**.

10. Close the **Controller Properties** window.

    **NOTE**: If you want to see the differences in the Compare Summary, Click the **Summary** Folder in the left column.
The next folder in the Summary column is labeled **Modules**.

11. Click on the **Modules** folder.

12. Expand the **Local** folder in the right project column.

13. This shows the I/O Configuration from the completed ACD file. The **EN2T** is **Red** signifying it does not exist in the L75_Project_Start ACD file.

14. Double-click on the **EN2T**.

The module’s full definition is displayed.

Remember! **Red** - Indicates items that exist in one project but not the other.

16. There are two other folders in the Summary window indicating other project differences: **Tasks** and **Logical Organizer View**. Using what you’ve learned above, click on each different folder and observe what is different.

17. Double-Click on **Tasks** and expand the sub-folders.

18. Double-Click on **Logical Organizer View** and expand the sub-folders.
3.3 Comparing Project Exports

New in v6.10 of the Logix Designer Compare Tool we can now compare project export files. This allows us to compare exported Programs, Routines, Rungs, AOIs, etc... In the example below we will step you through the comparison of two exported Programs. Then you will have to opportunity to compare routines, rungs, and AOIs.

19. Start a new comparison by clicking on File and then New in the top menu.

![Logix Designer Compare Tool window](image)

Note, our open project files path is:

Project Path: `C:\ Lab Files\Logix Advanced Lab`

20. For the **Left Content**, click the Browse button, navigate to the Lab3 Compare and Merge Tool folder and select the `Pump1.L5X` file using the path above.

21. For the **Right Content**, click the Browse button, navigate to the Lab3 Compare and Merge Tool folder and select the `Pump2.L5X` file using the path above.
Compare Options

For this lab we compared the Left and Right Content using only the tag values. But, you can also compare descriptions, constant tags, filter on tags or only include tracked components.

Tracked Components is a new feature in v31. In Studio 5000® Logix Designer you can configure components in your application to be tracked. These components will then be flagged whenever they are changed. The Logix Designer Compare tool, Studio 5000® Logix Designer and even the controller project can then check if the component has been changed.

22. Click OK to Start the compare.

23. From the Compare Results we have 1 Task Property Difference. Let’s dig deeper.
24. Click on Tasks in the Compare Summary.
25. Expand the Left Content and Right Content results.

26. Double Click on one of the blue Connections folders.

27. The results below show the differences between the two Programs in red text. These differences are all on the Program Parameter Connections. The black text in the middle of the results is not a change.

28. Double Click on the Unscheduled Programs / Phases folder.
29. This will display the changes for the Programs.

NOTE: The two Programs that we are comparing are shown under Unscheduled Programs / Phases because the compare files are Program Exports. Since the files that are being compared only contain the program data and not the information for the entire controller the Programs are Unscheduled. No Tasks have been configured to own these Programs.

![](image)

30. You can also compare other types of export files! The table below contains a list files that are available in the lab file folder that can be compared. Use this section of the lab to explore compare options.

NOTE: All files are in the **C:\Lab Files\ Advanced Logix \ Lab 1 Usability Enhancements** folder.

<table>
<thead>
<tr>
<th>Export Type to Compare</th>
<th>File Name</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>Pump1.L5X</td>
<td>Pump2.L5X</td>
</tr>
<tr>
<td>Routine</td>
<td>Tank1_MainRoutine.L5X</td>
<td>Tank2_MainRoutine.L5X</td>
</tr>
<tr>
<td>Rungs</td>
<td>Auto_Cycle1_Rungs0to3_from_MainRoutine.L5X</td>
<td>Auto_Cycle2_Rungs0to3_from_MainRoutine.L5X</td>
</tr>
<tr>
<td>AOI</td>
<td>AOI_ModuleSts_V01.L5X</td>
<td>AOI_ModuleSts_V02.L5X</td>
</tr>
</tbody>
</table>

NOTE: Not all of the files have differences! Some of the differences may be different than what you expect because these examples use Program Parameters.
3.4 Merging Project Files

Now that we have a comparison of the two files, we can invoke the new merge feature of Studio 5000 V24.

31. To start the merge, click on the New Merge icon in the toolbar.

![New Merge dialog box](image)

32. This will open up the New Merge dialog box.

![New Merge dialog box](image)

33. For the Left Content:, click the Browse button, navigate to the Lab3 Compare and Merge Tool folder and select the L75_Project_Start.ACD file using the path above.

34. For the Right Content: follow the same path and select the Line_Expansion.ACD file.

![New Merge dialog box](image)

35. Press OK
36. This will open up the comparison between the two projects.

NOTE: An Error window may be displayed. You can close it.

NOTE: Notice certain icon/ boxes are checked. Refer to the table below for icon definition.

NOTE: Blue text is from unresolved items. These items must be resolved to complete the Merge.

NOTE: Black is for resolved items. All items must be resolved to complete the Merge.

NOTE: Greyed out checked boxes indicate that the result of the merge will have items from both files.

37. The table below describes the meaning of the icons you will see in this comparison window.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>←→</td>
<td>Indicates the items are from the left project.</td>
</tr>
<tr>
<td>↔</td>
<td>Indicates the items are from the right project.</td>
</tr>
<tr>
<td>○</td>
<td>Indicates the items are from the center project.</td>
</tr>
<tr>
<td>☕</td>
<td>Indicates the items are from more than one projects.</td>
</tr>
<tr>
<td>blank</td>
<td>Indicates no differences are found for the current item.</td>
</tr>
</tbody>
</table>
38. Check the checkbox for Controller Tank_Controller in the left hand column.

39. Click on the right arrow next to the Tasks folder to expand the entry.

40. You will notice that a navigation aid exists in the toolbar in the form of the home icon, an arrow and ‘Tasks’. Clicking on the Home icon (house) will take us back to the original screen.

41. Check all three checkboxes in the left hand column for MainTask, unnamed middle check box, and Sim_Periodic.

42. Check the Sim_Logic checkbox in the right column.

43. Uncheck the Sim_Logic checkbox in the right column.

   NOTE: The unnamed box in the left hand column must be checked to have the Sim_Logix box in the right column unchecked. This unnamed box indicates that one of the tasks will not be merged. In this case it is Sim.Logic.

44. Double-click on MainTask

45. Verify the ScheduledPrograms Checkbox is checked in the left column.
46. Double-click on **ScheduledPrograms** in the left column.
47. Check the: **Pump2**, **Tank2** and **Auto_Cycle2** Checkboxes in the right column.

48. Click on the **Home** icon to navigate back home.
49. Double Click on the **Programs** folder.

   **NOTE:** The blue text is from unresolved items. These items must be resolved to complete the Merge.
50. Check the following check boxes in the Left Hand Column:
   i. Auto_Cycle1
   ii. Pump1
   iii. Tank1

51. Verify the following check boxes in the Right Hand Column are checked:
   i. Auto_Cycle2
   ii. Pump2
   iii. Simulation2
   iv. Simulation_Logic
   v. Station_1
   vi. Station_2
   vii. Tank2
   viii. Tank_Manager

52. Verify the items on the right hand side match the boxes shown below.
53. This configuration will keep the **Auto_Cycle1**, **Pump1** and **Tank1** from our **L75_Project_Start.ACD**. We will be combining these items with the additional items from **Line_Expansion.ACD** in our new merged project.

54. Navigate back home by clicking on the **Home** icon

55. Verify the items in the **Resulting Project** window. This is the bottom pane in the **Merge Tool**.

   NOTE: The Resulting Project window may be floating outside the Merge Tool window.

The representations for each of these icons in this lower pane are as follows.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![→]</td>
<td>Indicates the items are from the left project.</td>
</tr>
<tr>
<td>![←]</td>
<td>Indicates the items are from the right project.</td>
</tr>
<tr>
<td>![○]</td>
<td>Indicates the items are from the center project.</td>
</tr>
<tr>
<td>![⇒]</td>
<td>Indicates the items are from more than one project.</td>
</tr>
<tr>
<td>![blank]</td>
<td>Indicates no differences are found for the current item.</td>
</tr>
</tbody>
</table>

This pane shows which direction the merged items will be flowing. Using the key above, we can see that most of the items will be taken from the right pane. If you made some changes, you will see those indicated in the directional arrows. If we click, for instance on **Auto_Cycle1** we will see why the icon is the arrow.

NOTE: Greyed out checked box indicate that the result of the merge will have items from both files.
56. Now all of the items have black text and we are ready to save the project merge.

### 3.5 Saving Merged Project

57. Once the Merge project is set to your satisfaction, we can save it in any of 3 formats: **ACD**, **L5K**, or **L5X**.

58. Click the **Save** icon in the toolbar to begin the Save.

![Screenshot of the Logix Designer Merge Tool](image)

NOTE: If the window below is displayed your application still has merge items that you will need to resolve before the application merge results can be saved. Go back to **Step 39** and resolve all remaining items with blue text.

![Warning dialog box](image)

59. The **Save As** window will open.

![Save As window](image)
60. By default, the file will be saved as **MergeResult.ACD**, but you can change the name or the file type before saving.
   a. Also by default it will be stored in C:\Program Files (x86)\Rockwell Software\Logix Designer Tools\Logix Designer Compare Tool. You can change the save location if you would like.

61. Select Save to save the MergeResult.ACD file.

62. Once the merge process is complete a message will be displayed showing that the merge is complete with no error and no warnings.

![Error Window]

**NOTE:** Any errors or warnings in the new Merged ACD file will be show in the Error window.
**NOTE:** You must confirm all merged project information prior to deploying the project in a product environment. We will confirm the merged project in the following steps.

63. Click the **Open Merge Result** button on the toolbar.

![Open Merge Result Button]

64. The **Open Merge Result** button will open the merged project in Logix Designer.
3.6 Reviewing Merged Project

65. **Verify** the project. You may see some errors and warnings, but they are well documented and most likely the result of program connections that are missing. Notice the merged program has a **Tank2**, **Pump2**, and **Auto_Cyle2** Program along with an **EN2T Module** in the local chassis.

Note: Program Connections are described in another lab session.

![Diagram of Controller Organizer showing Tank2, Pump2, and Auto_Cyle2](image)

**Congratulations!**

You have just finished this Lab

[Click to Proceed to Contents](#)
Lab 4: Using AOI's to Monitor Module Status <15 minutes>

This section will provide an introduction to features that were introduced in version 24 of Studio 5000® Logix Designer. This feature is designed to extend the functionality of Logix Add On Instructions (AOI).

AOI Hardware Status:

- **Module as an InOut Parameter in an AOI**: The predefined data type Module can now be added as an InOut parameter to an AOI. When instantiating the AOI, the user can use a pull down of the I/O Configuration to select the module linked to the AOI. This gives the user a great method of linking a module to the logic inside of an AOI.

- **Module Class available inside an AOI**: GSV instructions can now access the Module Class inside of an AOI. This allows the AOI to monitor attributes such as: EntryStatus, FaultCode, LEDStatus and the new Path attribute.

- **Module Path Attribute**: The Module Class of a GSV instruction now has a new Attribute called Path. This returns the module path. The module path can be used in a message instruction to point the message directly at the module.

In this section, you will learn the benefits of these new features. Examples will be given of how they can be used in your next application. You will be using an existing program in an effort to conserve time.

4.1: Using Module as an InOut Parameter

This section will review an example using the Module predefined data type in an AOI. The example uses the Module predefined data type and retrieved the module's EntryStatus attribute. The EntryStatus attribute is analyzed to set descriptive status bits for the module.

1. Double-Click on the Lab Files folder on the desktop.

2. Navigate to the link below and double-click the project named Hardware_Monitoring_ModuleSts.ACD to open it in Logix Designer.

   You can find the project in this directory: C:\Lab Files\Advanced Logix\Lab 4 AOI Hardware Monitoring

3. Click on Communications => Who Active at the top of the screen.

   NOTE: RSLinx may take a few moments to open.
4. Expand the Ethernet/IP Driver **AB_ETHIP-1**.

5. Select **192.168.1.218, 1756-L85E**. You do not need to expand the controller.

6. Click **Download**.

7. When prompted, click **Download**.

8. If prompted to change the controller back to **Remote Run**, Click **Yes**.

9. If not prompted, changed the controller mode to **Remote Run** in Logix Designer.

10. In the Main Task, expand the program **CommSts_Local** and open the routine **Slot03_ModSts**
11. Monitor Rung 1. Notice in the **AOI_Module_Sts_V01**, the **IO_Mod_Ref** parameter uses the MODULE datatype (place your mouse over the Digital_Input tag). We are only able to pass the MODULE datatype into the AOI. The AOI_ModuleSts_V01 returns the running status of the module inputted in to the AOI.

![Diagram](image)

Note: The branch below the AOI_ModuleSts_V01 instruction is used to show the values of the output of the AOI without having to look them up in the Monitor Tags view.

12. To see the AOI work we will inhibit the Digital_Input module in slot 3.

14. Navigate to the Connection tab. Check Inhibit Module. Then Click on OK.

15. Press Yes on the Danger; Connection Interruption screen that will popup.
16. Navigate back to the **Slot03_ModSts** routine.

17. Now you can see that the **Local_03_ModSts.Inhibited** tag is **True** and the Local **Local_03_ModSts.Running** tag is **False**.

18. You can now remove the Inhibit from the **Digital_Input** module in slot 3 in the I/O Configuration.

19. Go back to the **Slot03_ModSts** Routine and verify that **Local_03_ModSts.Inhibited** tag is **False** and the Local **Local_03_ModSts.Running** tag is **True**.

20. Now, let’s take a closer look at the linking of the module into the AOI. This linking is done by selecting the module from the I/O Configuration.

21. Select Rung 1 of the Slot03_ModSts routine. Press the **Start Pending Rung Edits** button in the toolbar on top of the Ladder viewer/editor window.

22. Let's change the module that **IO_ModRef** in the **AOI_ModuleSts_V01** instruction is referencing.
23. Double click **Digital Input** in the AOI to get the pull down to appear.

You can browse to another module using the I/O Configuration tree to select the module. Any module can be selected. However, we can also choose to use program parameters of the Module datatype for code reuse.

a) Select the parameters **IO_LocalSlot_03**.

b) Open the Program Properties for **CommSts_Local** by right clicking on the program and selecting **Properties** from the drop down menu.

c) Select the **Parameters** tab and click **IO_LocalSlot03**. Now you can see the configuration of IO_LocalSlot03 and how it links to the module in the I/O Configuration named Digital_Input.

Note: Using this method we can have the same logic in many projects and just change the connection for each use!
24. Go to routine **Slot03_ModSts**.

25. Click the **Accept Pending Program Edits** button. Select **Yes** to the dialog that appears.

26. Click the **Test Accepted Program Edits** button. Select **Yes** to the dialog that appears.

![Accept Pending Program Edits]

![Test Accepted Program Edits]

In the **AOI_ModuleSts_V01** instruction of this rung you will notice that the parameter is used in place of the module name that was used in our first example.

![AOI_ModuleSts_V01]

27. Verify that this logic functions accordingly by inhibiting and un-inhibiting the module **Digital_Input** as you did earlier.

28. Click **Assemble Accepted Program Edits** on the toolbar. Click **Yes** to the dialog that appears.

![Assemble Accepted Program Edits]

29. Close Studio 5000 Logix Designer. If prompted to save the file respond with **No**.
4.2: Using the Module Object Path Attribute

This section will review an example of using the Path attribute of the Module Object. This example re-uses a single message instruction to return the module firmware revision from multiple modules. The message instruction performs a Device Who. This returns the same information that we see in RSLinx Classic when we right-click on a device in the RSWho screen and select Device Properties.

**Additional Information – Module Object Path Attribute**

**Example of Use**

The Path attribute of the Module object can be retrieved by using a GSV instruction. It returns the data shown below.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Instruction Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>SINT</td>
<td>Provides the CIP path to the module.</td>
</tr>
</tbody>
</table>

**Byte Meaning**

<table>
<thead>
<tr>
<th>Byte</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Length of the path in bytes. If 0, length of the SINT array is insufficient to hold the returned module path.</td>
</tr>
</tbody>
</table>

**Note:** If SINT array length is insufficient to hold the path, the array is zeroed out, and a minor fault is logged.

Since SINT array indexes 0 and 1 contain the length, some additional logic is required to copy the Path attribute output into the Message Datatype Path STRING member. The logic shown below from the AOI_ModWho_V01 uses this logic. This logic only uses byte 0 which limits us to a maximum of 255 characters for our length. This should be more than enough for a real world MESSAGE datatype Path.
The BTD copies the SINT path length from the SINT array to the LEN member of the String. The GSV returns SINT's, but the STRING datatype using a DINT for the LEN (length). Using the BTD we insure that the value is not converted. The COP instruction copies the remaining data to the Path DATA member – the string value of the message path. This time we are copying from SINT to SINT.

30. Double-Click on the Lab Files folder on the desktop.

31. Navigate to the link below and double-click the project named **Hardware_Monitoring_ModuleWho.ACD** to open it in Logix Designer application. You can find the project in this directory: C:\Lab Files\Advanced Logix\Lab 4 AOI Hardware Monitoring

32. Click on **Communications => Who Active** at the top of the screen.
   NOTE: RSLinx may take a few moments to open.

33. Expand the Ethernet/IP Driver **AB_ETHIP-1**.

34. Select 192.168.1.218, 1756-L85E. **You do not need to expand the controller.**

35. Click **Download**.

36. When prompted, Click **Download**.
37. If prompted to change the controller back to **Remote Run**, **Click Yes**.

38. If not prompted, changed the controller mode to **Remote Run** in Logix Designer.

39. In the **MainTask**, expand the program **ModuleWho_Local** and open the routine **Slot03_ModWho**.

![Diagram](image)

40. Monitor Rung 1. The **AOI_Module_Who_V01** retrieves the **Major Firmware** and **Minor Firmware** revisions of the module linked to the InOut Parameter **IO_ModRef**. On controller first scan all of the major and minor firmware revision tags are cleared.

   The AOI uses the new **Path** attribute of the MODULE object to point the message instruction to the module linked to **IO_ModRef**. This gives us an easy way to get module information. It also reuses a single message instruction, single message tag and associated code!

   The message instruction in the AOI uses the **DeviceWho** message. This returns more data than just the modules firmware revision. Reference Rockwell Automation KnowledgeBase Article 28917 for details on all of the data returned by the DeviceWho message.
41. Select the XIC instruction **Cmd_Local03_ModWho**. Right-click and select **Toggle Bit** from the menu or press Ctrl-T.

42. This triggers AOI_Module_Who to go out and retrieve the **Major_Firmware** and **Minor_Firmware** revisions from the linked module **Digital Input**. You will see the final values in the **Local03_Major_Firmware** and **Local03_Minor_Firmware** controller scope tags.
43. Open **RSLinx Classic** from the desktop and use **Communications/ RSWho** to browse to the **Digital Input** module (IO Module in Slot 3).

44. Right click on the input module and select **Device Properties**. From the screen that pops up verify the values you have in **Local01_Major_Firmware** and **Local01_Minor_Firmware** Logix controller scope tags.

**NOTE:** The Major and Minor firmware revisions of your modules may be different than shown but the values reported in Logix Designer and RSLinx Classic will align.
45. You can follow the same steps with the AOI in the Routine **Slot04_ModWho**. This AOI monitors the **Digital_Output** module in Slot 4.

![AOI Monitor Window](image)

46. Just like with the module in Slot 3, you can verify the results by bringing up the digital output module's **Device Properties** in RSLinx Classic.

47. Select **Go Offline** with the controller.

![Go Offline Button](image)

48. Close Logix Designer. If prompted to save the project respond with **No**.

**Congratulations!**
You have just finished this Lab

[Click to Proceed to Contents](#)
Lab 5: Improving Efficiency Using Partial Import Online <30 minutes>

This section will cover how to use some of the features that were introduced in version 24 of Studio 5000® Logix Designer to improve efficiency. We will be working in both the Controller and Logical Organizer windows. We will also be discussing the Partial Import Online (PIO) functionality.

5.1: Code Library Example

This section will demonstrate an example using an ACD file as an overall code library. We will start by opening two Studio 5000 .acd files.

1. Double-click on the Lab Files folder on the desktop.

2. Open the project Tank_Controller.ACD from the C:\Lab Files\Advanced Logix\Lab 5 Improving Efficiency Using Partial Import Online folder on the desktop.

3. Minimize the Tank_Controller project after it opens.

4. Go back to the Lab Files folder on the desktop.
5. Size the Logix Designer windows for Tank_Supervisor and Tank_Controller so they look like the screen shot below.

Let's assume that the Tank_Controller.ACD file is your company's code module repository. This essentially means the overall code library for your company is stored in an ACD file. For this example, we will assume you must roll out a tank control system. We created a new ACD file called Tank_Supervisor. You've opened both projects, and now it's time to drag and drop some of our code modules from our library to the actual Tank_Supervisor ACD file.

6. Make sure you select the Controller Organizer for each application.
7. Select the 1756_IB16IF module in the Tank_Controller.ACD file and drag / drop it into the Tank_Supervisor.ACD file. Make sure you drop it onto the 1756 backplane.

8. Select the rest of the I/O modules (1756_OB16IEF, 1756-IF8I and 1756-OF8) modules in the Tank_Controller.ACD file and drag / drop it into the Tank_Supervisor.ACD file in the same manner.
9. The I/O Configuration should look like the following screen shot in the **Tank_Supervisor.ACD** file.

![I/O Configuration](image)

10. Click on the **Logical Organizer** in both applications.

![](image)

11. Select the **Simulation_Logic** folder in the **Tank_Controller.acd** file and drag / drop it onto the **Logical Model Tank_Supervisor** in the **Tank Supervisor.acd**.
12. Select the **Tank_Manager** folder in the **Tank_Controller.acd** file and drag / drop it into the Logical Organizer for the **Tank_Supervisor**.

![Logical Organizer](image)

**NOTE:** It is extremely easy and fast to drag and drop items between instances of Logix Designer. Many users will store their code libraries in an ACD file for this purpose. The introduction of the Logical Organizer gives the user a much better visual representation of a system or process and its components from a modular view.

13. We are finished with the **Tank_Controller.acd** at the moment. Minimize the **Tank_Controller.acd** file, but do not close it.

14. Maximize the **Tank_Supervisor.acd** file.

**NOTE:** When you drag and drop items into the Logical Organizer, a “deep” copy will be performed. This means that when a folder or program is copied, all sub-folders and sub-programs are copied. Additionally, when you drag and drop items into the Logical Organizer, programs are not scheduled. We must now schedule the new programs.

15. Click on the **Controller Organizer** in the **Tank_Supervisor.acd** file.

16. All programs should be in the “**Unscheduled Programs / Phases**” folder. Let’s create the Periodic Task for the Simulation Logic.
17. Right-click on the **Tasks** folder and select **New Task**…

![Controller Organizer](image)

18. Enter **Sim.Logic** into the name. Leave everything else at the default value, click **OK**.

![New Task](image)

19. Drag and drop the **Simulation1** Program into the **Sim.Logic** Task.
20. Drag and drop the **Pump1** Program into the **MainTask** Task.

21. Drag and drop the **Tank1** Program into the **MainTask** Task.

22. Drag and drop the **Auto_Cycle1** Program into the **MainTask** Task.

![Controller Organizer Diagram](image)

**NOTE:** This task/program structure should look nearly identical to the file we used in Sections 1 and 2. You created this one from scratch extremely quickly using an ACD file that contains the same code modules (programs). This is one major advantage to using modular coding design practices.

23. Click on **Communications => Who Active** at the top of the screen.

24. Expand the Ethernet/IP Driver **AB_ETHIP-1**.
25. Select **192.168.1.218, 1756-L85E** You do not need to expand the controller. Press **Download**.

26. When prompted, press **Download**.

27. When prompted to change the controller back to **Remote Run, Click Yes**.

28. Verify the application works as expected.

29. Make sure the switch at **DI4** is turned to the **RIGHT**. This switch allows the user to select between Auto (Right) and Manual (Left) Mode.

30. When the green light at **DO0** is illuminated, the Fill Pump is ON.

31. When the light at **DO2** is illuminated, the Drain Pump is ON.

32. The Voltmeter at **AO0** simulates the tank level. 0 volts indicates EMPTY, 10 volts indicates FULL.

33. Turn the **A10 Knob to 5**. The potentiometer knob simulates the pump speed. As you turn the knob clockwise, the pump speed will increase.
   
   **NOTE:** You should notice the tank slowly filling and draining. As the voltmeter swings from left to right, the **DO0** light should be illuminated (filling). As the voltmeter swings from right to left, the **DO2** light should be illuminated (draining).

34. Turn the **A10 Knob to 10**.
   
   **NOTE:** The rate at which the tank is filling and draining should have increased.

35. Turn the **DI4** switch to the **LEFT** position. This will put the system into Manual Mode.
   
   **NOTE:** The lights should both turn off. The voltmeter should stop moving. Since the system is now in manual mode, you must manually operate the fill and drain pumps.

36. Press and HOLD the green button at **DI0**. This will manually turn the fill pump ON.
   
   **NOTE:** You should notice the green light will turn ON and the voltmeter will be increasing. When the tank is full, the green light should turn off even if the button remains depressed.

37. Press and HOLD the green button at **DI2**. This will manually turn the drain pump ON.
   
   **NOTE:** You should notice the light will turn ON and the voltmeter will be decreasing. When the tank is empty, the green light should turn off even if the button remains depressed.

38. **Go Offline** with the controller.

![Go Offline with the controller](image.png)
5.2: Line Expansion Example

This section will demonstrate an example of using modular programming techniques to simulate a line expansion. Let’s assume this application was already in operation in the field. During the design phase, we knew we would be adding a second Tank Station after the initial startup. With that in mind, we made sure to include enough I/O to support the expansion. Let’s walk through the process of adding the second Tank Station.

**Partial Import Online (PIO)**

PIO brings even more power and flexibility to the design environment when implementing Program Parameters. You have the flexibility to store all code modules (programs / AOIs) in a dedicated library ACD file or as exported xml files. When these files are imported or copied into the ACD file, you will have a detailed utility (GUI) that will allow the user to view / handle collisions, rename items, and most importantly configure connections. PIO will allow you to import these programs while offline OR online, while giving you quick access to all of the program’s parameters and connections prior to the import.

39. Click on the **Logical Organizer** in the **Tank_Supervisor.acd** file.

40. Expand the **Simulation.Logic** folder.
41. **Right-click** on the **Simulation Logic** folder. Select **Add => Import Program**.

   **NOTE:** Additionally, you can left-click drag and drop components which will function similarly to copy/paste. Using right-click drag and drop of components will give you a few more options on how the component is pasted / imported.

42. Navigate to **C:\Lab Files\ Advanced Logix \ Lab 5 Improving Efficiency Using Partial Import Online\Library**.

43. Select the **Simulation.L5X** file.

   **NOTE:** This file is an export of the Simulation_1 program. We will be using the PIO functionality to import this program into our application.

44. Click the **Open** button.
45. The **Import Configuration** window will be displayed.

![Import Configuration window](image)

**NOTE:** This window allows us to see the details about the import. We can view and correct collisions as well as make changes to connections.
46. Verify the **Simulation1** program in the “Import Content” window is highlighted.

![Image of Import Content window with Simulation1 highlighted]

NOTE: If a red flag appears in the yellow margin it indicates that an item is flagged for review. The user should investigate all flags prior to the import.

47. In the **Configure Program Properties** window, change the Final Name to **Simulation2**.

![Image of Configure Program window with Final Name changed]

48. Click in the **Description** box. Clicking in a different box will accept the name change.

NOTE: Notice how the operation changed to “Create”.

![Image of Configure Program window with Final Name changed and Operation set to Create]
49. In the “Schedule In” drop-down, verify it is selected as Sim.Logic.

![Schedule In dropdown](image)

NOTE: This will schedule the Simulation2 program in the Sim.Logic Periodic Task.

50. The Configure Program Properties window should look like the screen shot below.

![Configure Program Properties](image)

51. Click on the Connections folder in the Import Content window.

![Connections folder](image)
52. In the **Configure Connections** window, scroll to the right to view the **Final Connection** column.

<table>
<thead>
<tr>
<th>Usage</th>
<th>Input Connection</th>
<th>Operation</th>
<th>Final Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>\Pump1.Out_Drain</td>
<td>Connect</td>
<td>\Pump1.Out_DrainPumpON</td>
</tr>
<tr>
<td>Input</td>
<td>\Pump1.Out_Fill</td>
<td>Connect</td>
<td>\Pump1.Out_FillPumpON</td>
</tr>
<tr>
<td>Input</td>
<td>\Pump1.Val_Pump</td>
<td>Connect</td>
<td>\Pump1.Val_PumpSpeed</td>
</tr>
<tr>
<td>Output</td>
<td>\Tank1.Val_Tank</td>
<td>Connect</td>
<td>\Tank1.Val_TankLevel</td>
</tr>
</tbody>
</table>

NOTE: This window will allow you to create new connections, change the connections on the imported tags, or disconnect connections. For this example, you will notice all connections are tied to Pump 1 and Tank 1. Since we have not imported Station 2, we should disconnect these four connections. They will be properly connected on the import of Station 2.

53. Click in the **Operation** box and select the drop-down. Select **Disconnect** for each connection.

54. Click **OK**.

55. The **Simulation.Logic** folder should now have a new program called **Simulation2**.

Now it’s time to import the station programs for station 2.

56. Expand the **Tank.Manager** folder.
57. **Right-click** on the **Tank_Manager** folder. Select **Add => Import Program**.

**NOTE:** Additionally, you can left-click drag and drop components which will function similarly to copy/paste. Using right-click drag and drop of components will give you a few more options on how the component is pasted/imported.

58. Navigate to `C:\Lab Files\Advanced Logix \ Lab 5 Improving Efficiency Using Partial Import Online\Library`. Select the **Station.L5X** file.

**NOTE:** This file is an export of the Station_1 folder. We will be using the PIO functionality to import this program into our application.
59. Click the **Open** button.

60. The **Import Configuration** window will be displayed. It should look like the screen shot below.

![Image of Import Configuration window]

61. In the **Configure Program Properties** window, change the **Final Name** to **Station_2**.

![Image of Configure Program Properties window with Final Name set to Station_2]

62. Click in the **Description** box to accept the name change. The Operation now is automatically changed to Create.

![Image of Configure Program Properties window with Operation set to Create]

63. In the "**Schedule In**" drop-down, select **MainTask**.

![Image of Schedule In drop-down set to MainTask]

**NOTE**: This will schedule all programs in the Station_2 folder in the MainTask.
64. The **Configure Program Properties** window should look like the screen shot below.

![Configure Program Properties screenshot]

65. Click on the **Programs** folder in the **Import Content** window.
66. In the **Final Name** column, make the following changes:

- Auto_Cycle1 to **Auto_Cycle2**
- Pump1 to **Pump2**
- Tank1 to **Tank2**

![Programs Table](image)

67. Click on the **Connections** folder in the **Import Content** window.

![Connections Folder](image)

This is where the PIO feature can really start to improve efficiency. The Configure Connection window will allow you to make changes to connections in the parent program as well as all children programs that are within the parent program.

In this example, the Station_2 folder is the parent and the Pump2, Tank2, and Auto_Cycle2 programs are children. We will have access to all connections from a single window. Since Station_2 must control an entirely different set of hardware, we must make the applicable changes to several connections. The following steps will walk you through this process.
68. Widen the **Import Configuration** window and expand the **Final Connection** column so you can see the **Parameter Name** and the **Final Connection** columns.

![Import Configuration window](image)

69. Change the Final Connection for **\Pump2.Out_DrainPumpON** to **\Simulation2.Inp_DrainPumpON**.

![Configure Connections table](image)

**NOTE:** You will notice that this parameter is listed twice in the list. This is normal when “fanning” is implemented. Fanning is when multiple connections are configured to one parameter.

70. Change the Final Connection for **\Pump2.Out_FillPumpON** to **\Simulation2.Inp_FillPumpON**.

![Configure Connections table](image)

**NOTE:** You will notice that this parameter is listed twice in the list. This is normal when “fanning” is implemented. Fanning is when multiple connections are configured to one parameter.
71. Change the Final Connection for \Pump2.Val_PumpSpeed to \Simulation2.Inp_PumpSpeed.

72. Change the Final Connection for \Tank2.Val_TankLevel to \Simulation2.Out_TankLevel.

We have a few more I/O tags to change as well.

73. Change the Final Connection for \Pump2.Inp_FillPumpPB to Local:3:I.Pt[1].Data.


75. Change the Final Connection for \Pump2.Inp_AutoMode to Local:3:I.Pt[5].Data.

76. Change the Final Connection for \Auto_Cycle2.Inp_ModeSwitch to Local:3:I.Pt[5].Data.


NOTE: You will notice that these parameter is listed twice in the list. This is normal when “fanning” is implemented. Fanning is when multiple connections are configured to one parameter.

NOTE: We are not changing the connection for \Pump2.Inp_PumpSpeed. We will connect to the same POT as Station_1.

79. Change the Final Connection for \Tank2.Pv_MSG_PLC5Read to MSG_PLC5_Read_Station_2.
80. Change the Final Connection for \Tank2.Inp_LeakDetectionPLC5 to PLC_LeakStatus_Tank2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data Type</th>
<th>Usage</th>
<th>Import Connection</th>
<th>Operation</th>
<th>Final Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tank2.ValTank2</td>
<td>INT</td>
<td>Input</td>
<td>PLC5_LeakStatus_Tank1</td>
<td>Connect</td>
<td>Tank2.Inp_LeakDetectionPLC5</td>
</tr>
</tbody>
</table>

81. The **Final Connection** column should look like the screenshot below.

NOTE: Verify that all final connections match the screenshot below. The red arrows indicate the connections that have been changed.

NOTE: The tag MSG_PLC5_Read can only have one connection. You will need to create a new Message tag to connect to the 'Tank2.Pv_MSG_PLC5Read' parameter. The new message tag is not shown.

82. Click **OK**.

83. The **Tank_Manager** folder should now have a new folder called **Station_2**. Expand the **Station_2** folder and verify that the programs imported successfully.
84. Click on **Communications => Who Active** at the top of the screen.

![Image](https://via.placeholder.com/150)

85. Expand the Ethernet/IP Driver **AB_ETHIP-1**

![Image](https://via.placeholder.com/150)

86. Select **192.168.1.218, 1756-L85E**. You do not need to expand the controller. Press **Download**.

87. Select **192.168.1.218, 1756-L85E**. You do not need to expand the controller. Press **Download**.

88. When prompted, press **Download**.

89. When prompted to change the controller back to **Remote Run, Click Yes**.

Verify the application works as expected for the new Station 2.

**NOTE:** Station 1 and Station 2 will be running at the same time. The will share the Pump Speed **AI0 Knob** and Tank Level **AO0 Voltmeter**.

A possible result with this logic is that the **AO0 indicator** can be “jumpy” even with both mode switches (**DI4 and DI5**) in manual mode. If you notice that the **AO0 indicator** is not moving smoothly over the range in both filling and draining, this is expected.
90. Make sure the switch at DI5 is turned to the **RIGHT**. This switch allows the user to select between Auto (Right) and Manual (Left) Mode for Station 2.

91. When the light at DO1 is illuminated, the Fill Pump is ON.

92. When the light at DO3 is illuminated, the Drain Pump is ON.

93. The Voltmeter at AO0 simulates the tank level. 0 volts indicates EMPTY, 10 volts indicates FULL.

94. Turn the A10 Knob to 5. The potentiometer knob simulates the pump speed. As you turn the knob clockwise, the pump speed will increase.

   **NOTE:** You should notice the tank slowly filling and draining. As the voltmeter swings from left to right, the green light should be illuminated (filling). As the voltmeter swings from right to left, the red light should be illuminated (draining).

95. Turn the A10 Knob to 10.

   **NOTE:** The rate at which the tank is filling and draining should have increased.

96. Turn the DI5 switch to the **LEFT** position. This will put the system into Manual Mode.

   **NOTE:** The lights should both turn off. The voltmeter should stop moving. Since the system is now in manual mode, you must manually operate the fill and drain pumps.

97. **Press and HOLD** the green button at DI1. This will manually turn the fill pump ON.

   **NOTE:** You should notice the green light will turn ON and the voltmeter will begin to increase. When the tank is full, the green light should turn off even if the button remains depressed.

98. **Press and HOLD** the green button at DI3. This will manually turn the drain pump ON.

   **NOTE:** You should notice the green light will turn ON and the voltmeter will be decreasing. When the tank is empty, the green light should turn off even if the button remains depressed.

99. **Go Offline** with the controller.

   ![Controller Interface](image)

100. Close Studio 5000 Logix Designer. If prompted to save the file respond with **No**.
5.3: PIO Search and Replace Example

This section will perform some of the same changes that we made in section 2, but we will use the Find/Replace features in the PIO window.

PIO Find and Replace

Using Find and Replace in the PIO allows you to quickly change many tags at one time. PIO will allow you to import these programs while offline OR online, while giving you quick access to all of the program’s parameters and connections prior to the import.

1. Double-Click on the Lab Files folder on the desktop.

2. Navigate to C:\Lab Files\Advanced Logix\Lab 5 Improving Efficiency Using Partial Import Online

3. Open the project named Tank_Supervisor_Section3.ACD.

4. Click on the Logical Organizer in the Tank_Supervisor_Section3.acd file.

5. Expand the Simulation Logic folder.
6. **Right-click** on the **Simulation.Logic** folder. Select **Add => Import Program**.

   NOTE: Additionally, you can left-click drag and drop components which will function similarly to copy/paste. Using right-click drag and drop of components will give you a few more options on how the component is pasted / imported.

    ![Image 1 of 6](image1.png)

7. Navigate to **C:\Lab Files\Advanced Logix\Lab 5 Improving Efficiency Using Partial Import Online\Library**. Select the **Simulation.L5X** file.

   NOTE: This file is an export of the Simulation_1 program. We will be using the PIO functionality to import this program into our application.

    ![Image 2 of 6](image2.png)

8. Click the **Open** button.
9. The **Import Configuration** window will be displayed. It should look like the screen shot below.

![Import Configuration Window](image)

**NOTE:** This window allows us to see the details about the import. We can view and correct collisions as well as make changes to connections.

10. In the **Configure Program Properties** window, change the **Final Name** to **Simulation2**.

![Configure Program Properties Window](image)

11. Click in the **Description** box. Clicking in a different box will accept the name change.

**NOTE:** Notice how the operation changed to “Create”. 

![Configure Program with Parenting Information](image)
12. In the “Schedule In” drop-down, select Sim_Logic.

![Image of Schedule In dropdown]

NOTE: This will schedule the Simulation2 program in the Sim_Logic Periodic Task.

13. The Configure Program Properties window should look like the screen shot below.

![Image of Configure Program Properties]

14. Click on the Connections folder in the Import Content window.

![Image of Import Content window with Connections folder highlighted]

15. Select Find/Replace...
16. Enter 1. in the Find What: box. (Be sure the Use Wildcards check box is not selected).

17. Enter 2. in the Replace With: box.

18. Click **Find Next**.

**NOTE:** In this example the Find / Replace has found the “1.” In the Final Connection \Pump1.Out_DrainPumpON

19. Click **Replace**

20. Click **Replace** 3 more times to replace all of the “1.” With “2.”.

21. Click **Close** to close the Find / Replace window

22. Click **OK** to finalize the import of Simulation2
23. The **Simulation Logic** folder should now have a new program called **Simulation2**.

![Logical Organizer diagram](image)

Now it’s time to import the station programs for station 2.

24. Expand the **Tank_Manager** folder.

![Logical Organizer diagram](image)
25. **Right-Click** on the **Tank_Manager** folder. Select **Add => Import Program**.

![Image of Tank_Manager folder with Add and Import Program options highlighted](image)

26. Navigate to **C:\Lab Files\Advanced Logix\Lab 5 Improving Efficiency Using Partial Import Online\Library**. Select the **Station_Section3.L5X** file.

   NOTE: This file is an export of the Station_1 folder. We will be using the PIO functionality to import this program into our application.

27. Click the **Open** button.

28. In the **Configure Program Properties** window, change the **Final Name** to **Station_2**.

![Image of Configure Program with Parenting Information dialog box](image)

29. Click in the **Description** box to accept the name change.

30. In the “**Schedule In**” drop-down, select **MainTask**.

   NOTE: This will schedule all programs in the Station_2 folder in the MainTask.
31. The **Configure Program Properties** window should look like the screen shot below.

![Configure Program Properties Window](image)

32. Click on the **Programs** folder in the Import Content window.

![Programs Folder](image)

This is where the PIO feature can really start to improve efficiency. The Configure Programs window will allow you to rename and/or create new Programs for your imported logic. In this example, we are renaming and creating new Programs at the same time before the new logic is imported.
33. The **Import Configuration – Programs** window will look like the screen shot below.

![Image of Import Configuration – Programs window]

34. Select **Find/Replace…**

![Image of Find/Replace dialog box]

35. Enter 1 in the **Find What:** box.

36. Enter 2 in the **Replace With:** box.

37. Click **Find Next.**

![Image of Find/Replace with highlighted row]

**NOTE:** In this example the Find / Replace has found the “1.” In the Final Name Auto_Cycle1
38. Click Replace

39. Click Replace 2 more times to replace all of the 1 With a 2 in the Program Section.

   NOTE: Only Click Replace a total of 3 times and only for the tags in the Program Section.

   The Find / Replace has now moved the screen to the Tags section. We will be very selective about which tags we replace in this section.

40. Click Replace 2 times to Replace the PLC5 Tags.

41. The Find / Replace has now moved the screen to the Connections section. We will be very selective about which tags we replace in this section.
42. Click **Replace 4** times to Replace the Simulation Tags.

43. Verify that your **Connections** sections matches the screen below.

This is where the PIO feature can really start to improve efficiency. Find and Replace and quickly change many tags before they are imported into your application. Tag changes can be made without typing or editing logic.

You can verify all of the tags and their connections before the logic is imported into your application! This saves time in reviewing and editing logic. You can be confident that your online logic import is correct before you hit the OK button that start the import.
44. Click **Close**

45. Manually update the I/O Tags in the **Connections** Section to match the I/O shown below:

![Configure Connections Table](image)

46. Click **OK**.

47. In the **Logical Organizer** the **Tank_Manager** folder will now have a new folder called **Station_2**.

48. Expand the **Station_2** folder and verify that the programs imported successfully.
49. In the **Controller Organizer** new programs have been added for Auto_Cycle2, Pump2, Tank2 and Simulation2.

![Controller Organizer Diagram]

**NOTE**: Station_2 will operate exactly like Station_1, only using different I/O and tags.

**PIO – You can quickly change names before importing!**

The PIO dialog window guides you through the names in the import file and the new names that are going to be used in the controller. You can quickly and easily change these names before anything is added to the controller!

**PIO – Partial Import Online - Offline and Online workflows are the same!**

You can complete the same steps while online with a running controller. The workflow is the same. The changes can be automatically accepted or added into test mode.

Let's download the application and verify Station_2 operates as expected.

Click on **Communications => Who Active** at the top of the screen.
50. Expand the Ethernet/IP Driver **AB_ETHIP-1**

51. Select **192.168.1.218, 1756-L85E**. You do not need to expand the controller. Press **Download**.

52. When prompted, press **Download**.

53. When prompted to change the controller back to **Remote Run**, Click **Yes**.

54. Make sure the switch at **DI5** is turned to the **RIGHT**. This switch allows the user to select between Auto (Right) and Manual (Left) Mode.

55. When the green light at **DO1** is illuminated, the Fill Pump is ON.

56. When the green light at **DO3** is illuminated, the Drain Pump is ON.

57. The Voltmeter at **AO0** simulates the tank level. 0 volts indicates **EMPTY**, 10 volts indicates **FULL**.

58. Turn the **AI0** Knob to **5**. The potentiometer knob simulates the pump speed. As you turn the knob clockwise, the pump speed will increase.

   NOTE: You should notice the tank slowly filling and draining. As the voltmeter swings from left to right, the green light **DO1** should be illuminated (filling). As the voltmeter swings from right to left, the **DO3** light should be illuminated (draining).

   NOTE: In this example the 2 stations are sharing the Pump Speed Input from **AI0** and the Tank Level Output from **AO0**. This is not ideal and in a real world application these would be different.

59. Turn the **AI0** Knob to **10**.

   NOTE: The rate at which the tank is filling and draining should have increased.

60. Turn the **DI5** switch to the **LEFT** position. This will put the system into Manual Mode.

   NOTE: The Green lights should both turn off. The voltmeter should stop moving. Since the system is now in manual mode, you must manually operate the fill and drain pumps.

61. **Press and HOLD** the green button at **DI1**. This will manually turn the fill pump ON.

   NOTE: You should notice the green light will turn ON and the voltmeter will being to increase. When the tank is full, the green light should turn off even if the button remains depressed.
62. **Press and HOLD** the red button at **DI3**. This will manually turn the drain pump ON.

   **NOTE:** You should notice the green light will turn ON and the voltmeter will being to decrease. When the tank is empty, the light should turn off even if the button remains depressed.

63. Close Logix Designer. If prompted to save the file respond with **No**.

   **Congratulations!**

   You have just finished this Lab

   [Click to Proceed to Contents]
Lab 6: Introduction to Program Parameters <40 minutes>

Since applications often consist of many code modules, each module must have a means of interacting with other modules or devices. This interaction is often referred to as the Linking Layer. Code Modules are linked together using clearly defined “inputs” and “outputs” that are required for a module to complete its task. Once a code module’s inputs and outputs have been defined, they can be linked to other modules using Connections.

Within the Logix environment, users can implement modular design techniques in a number of ways including the use of Routines, Programs, and AOIs. Execution varies depending on the type of container used, i.e. scheduled (Programs) versus called (AOI’s and Routines). Prior to version 24, if the application was segregated using programs, users were forced to use controller scoped tags for the linking layer. Version 24 introduces a feature called Program Parameters. This feature allows the user to configure program-to-program connections, without the use of controller scoped tags. The ability to define program-to-program interfaces adds an additional level of Encapsulation for an improved modular programming environment.

This section will provide an introduction to several powerful features introduced in version 24 of Logix Designer. These features are designed to provide a design environment that enables users to fully implement modular programming methodologies.

Program Parameter types:

Program parameters define a data interface for Programs. Data sharing between Programs can be achieved either through pre-defined connections between parameters or directly through a special notation. Unlike local tags, all program parameters are publically accessible outside of the program. Additionally, HMI external access can be specified on individual basis for each parameter.

There are two types of parameters for Programs: standard parameters, which are Input and Output, and special usage parameters, which are InOut and Public.

- **Input parameters**: an Input parameter defines the data that is passed by value into the executing program. Because Input parameters are passed by value, their values cannot change from external sources during the execution of the program when connected to Output or Public parameters. An Input parameter supports a maximum of one connection or InOut binding.

- **Output parameters**: an Output parameter defines the data that is produced as a direct result of executing the program. Because Output parameters are always passed by value, their values cannot change outside of the execution of the program nor can they be changed from other programs. Output parameters can be connected to one or several Input and Public parameters or Controller scope tags.

- **InOut parameters**: an InOut parameter represents a reference to data that can be used both as input and output during the execution of a program. Because InOut parameters are always passed by reference, their values can change from external sources during the execution of the program. An InOut parameter must be bound to exactly one target including Input and Public parameters as well as Controller scope tags. Some restrictions apply when binding an InOut to an Output parameter.

- **Public parameters**: a Public parameter defines the data that is passed by value into a program or
is produced by the executing program. A Public parameter accepts several connections or InOut bindings (with the exception of Controller scope tags) and can function as a data concentrator or shared memory between programs.

In this section, you will learn how connections are made between code modules (Programs). We will cover existing and new methods. We will primarily focus on how to create and connect Program Parameters (New in version 24). You will be using an existing program in an effort to conserve time.

6.1: Input Parameters

This section will cover Input parameters. The example application contains three code modules (Pump, Tank, and Auto_Cycle). The code module interface has been defined and parameters are already created. This was done to conserve time. In this section, you will be setting up several connections to Input parameters.

Goals:

- Create an Input Parameter
- Configure a connection
- Understand when an Input connection can be used

1. Double-Click on the Lab Files folder on the desktop.

2. Navigate to the link below and double-click the project named Sample_Project_Start.ACD to open it in Logix Designer.
   
   You can find the project in this directory: C:\Lab Files\Advanced Logix \ Lab 6 Introduction to Program Parameters

3. In the Controller Organizer, expand the Pump1 Program.
4. Double-click the **Parameters and Local Tags**

![Diagram showing Parameters and Local Tags]

**NOTE:** You will notice that Program Tags are now Parameters and Local Tags. This change was required to reflect the addition of Program Parameters. Additionally, the naming convention used for Programs in version 24 now matches the naming convention used for Add-On Instructions (AOIs).

5. Highlight the tag **Inp_DrainPumpPB**. This can be done by pressing the gray button to the left of the tag name. You may need to expand the name column.

![Tag table with Inp_DrainPumpPB highlighted]
6. Locate the tag properties window on the right side of the screen. If **Parameter Connections** is not expanded, do so now by clicking on the arrow symbol.

![Parameter Connections](image)

**NOTE:** A Parameter must be "connected" to another Parameter or Controller Tag. For example, connecting an Output parameter to an Input parameter will ensure that data is copied from the Output parameter to the Input on every scan. The screen shot above indicates that this input parameter has no connections. The steps below will walk you through creating a connection.

7. Click in the box that says **'New Connection'**.

![New Connection](image)

This input parameter is for the **Drain Pump Pushbutton** input. It will be connected to the physical input point in the Controller Tags.

8. Once you click in the box, click the gray browse button on the right.

![Parameter Connections](image)

9. Click the drop down arrow when the tag selection pop-up appears.
10. Locate and expand **Local:3.I.Pt**

![Image](image1.png)

NOTE: There are many ways to filter or search through tags on this screen. Take a moment to familiarize yourself with the many options. We will explore some of them as we progress through the lab.

11. Select **Local:3.I.PT[2]**. Click the symbol to expand the **Local:3.I.PT[2]** tag.

![Image](image2.png)

12. Select **Local:3.I.PT[2].Data**.

![Image](image3.png)
13. Once you select the Data bit, the tag selection screen should look like the screen shot below.

![Tag Selection Screen](image)

NOTE: If you know the exact tag name, you can manually enter the text in step 7. This can conserve time, but may become cumbersome as tag names grow in length and complexity.

14. Press the **OK** button. You should have the following:

![Parameter Connections Screen](image)

15. Notice how the \(0:0\) next to Parameter Connections changed to \(1:0\).

The number on the left between the brackets indicates the number of connections to the parent tag. In this case, the parent tag is *Inp_DrainPumpPB*. The number on the right indicates the number of connections to members of the parent tag. For example, if the parent tag was a User-Defined data type, then any connection to a member of the tag will be reflected in the number on the right. This also applies to connections that are configured to bits in a SINT, INT or DINT.

We need to create one more Input connection. Let’s create this connection by adding a connection column in our tag database.

16. Click on the **Edit Tags** tab for the Pump1 Parameters and Local Tags.
17. Right-click on any one of the column headers. Select **Toggle Column** and then select **Connection** if not checked.

![Right-click on column header](image)

**NOTE:** The user can customize the tag database columns. By default, the Connection column is not displayed.

18. Locate the tag **Inp_FillPumpPB**.

19. Click in the **Connection** box for this tag. (You may have to scroll to the right to see the **Connection** column).

![Connection box](image)

**NOTE:** At this point, you can click the gray browse button. This button will display the Connection Configuration screen. This window would be required if more than one connection exists or must be configured. If only one connection exists or is desired, you can click the drop down arrow. This will display the tag selection pop-up. You can also just type the name of the connection tag into the connection box. We are going to configure this connection through the Connection Configuration Screen.

20. Click on the browse button to open the **Connection Configuration Dialog Box**.
21. Click in the box with the text *New Connection.*

![Connection Configuration window](image)

**NOTE:** From this screen, you can either manually type the connection into the name box or you can use the tag selection pop-up. This example will use the pop-up.

22. Click on the gray button on the right side of the name box.

![Gray button](image)

23. Navigate to **Local:3.I.Pt[0].Data**. Click **OK**.

![Select Tag window](image)
24. Click **OK** to close the Connection Configuration Dialog Box.

25. Verify you have the following:

![Image of a table with parameters](image)

**Additional Information – Input Parameters**

**When to use Input Parameters**

Input Parameters can be connected to Controller Scoped tags, Output Parameters, Public Parameters, and InOut Parameters. This makes input parameters a great option for connecting input card data to a code module or object. It also is a good option for code module or object commands that are point-to-point. If multiple code modules or objects must have access to the command, there may be better options available and they will be discussed later in the lab.

**Input Parameter General Information**

- **Input parameters (including members) can only support ONE connection.** For example, let's assume you have a DINT input parameter in Program A named Input1. Let's also assume that Program B has a BOOL output parameter named Output1 that is connected to bit 0 of Input1 in Program A. An error will occur if any additional connection is defined for Input1 at the DINT level. The same rule applies for User-Defined Structures.

- **Input Parameter values are refreshed BEFORE each scan of a program.** The values will not change during the logic execution. Users will no longer have to write code to buffer inputs if they don't want them to change during program execution.

- **A program can write to its own input parameters.**
6.2: Output Parameters

This section will cover Output parameters. The example application contains three code modules (Pump, Tank, and Auto_Cycle). The code module interface has been defined and parameters are already created. This was done to conserve time. In this section, you will be setting up several connections to Output parameters.

Goals:
- Create an Output Parameter
- Configure a connection
- Understand when an Output connection can be used

26. In the Controller Organizer, right-click on the Tank1 Program and select Properties.

NOTE: There are several ways to configure connections. In the previous section, we used the Tag Properties window. In this section we will use the Parameters Tab in the Program Properties window. Both methods are available and can be used based on user preference.
27. Select the **Parameters Tab**.

NOTE: The Parameters tab is new with version 24. This gives the user another way to see and configure parameters and connections. (Do not worry about the Red “X” next to Pv_MSG_PLC5Read. We will discuss this later in the lab).
28. Click in the Name box for the tag **Out_VoltMeter**. (DO NOT click on the graphic. See the note below for more details.)

29. In the Connections Window at the bottom of the screen, in the Name box the text **New Connection**, type the following: **Local:6:O.Ch[0].Data.**

30. Click **Apply**.

31. Notice, there is a red X on the Out_VoltMeter. This is because we need to change the parameter data type to match our connection. Change the datatype for **Out_VoltMeter** to **REAL** and click **Apply**.

32. Click **OK** to close the Program Properties window.

NOTE: For this application, the Auto_Cycle code module must be able to command the Drain and Fill pumps to run. The Pump code module has been configured to have a public run command for both the Drain and Fill pumps. Since the Auto_Cycle module must be connected to the Pump module, an output parameter must also be configured for both the Drain and Fill pumps in the Auto_Cycle module. We will be configuring these connections as well as explore some of the filter / search mechanisms throughout the next several steps.
33. In the **Controller Organizer**, expand the **Auto_Cycle1 Program**.

34. Double-click the **Parameters and Local Tags**

So far, all connections that have been configured have been to Controller Tags (I/O). The next two connections will be connections between two programs.

35. Highlight the parameter **Out_DrainPump**. In the tag properties window, expand **Parameter Connections**.

36. Click in the box that says ‘**New Connection**’.

37. Once you click in the box, click the gray browse button on the right.

38. Click the drop down arrow when the tag selection pop-up appears.
39. Remove the checkbox on “Show Controller Tags”

40. Remove the checkbox on “Show Program Tags”

41. In the “Show Parameters from other Program” dropdown, select Pump1.

![Image of parameter list]

NOTE: Your window should appear like the screen shot above. You are only viewing the Pump1 parameters that can be connected to the Output parameter. All local tags and parameters that would make invalid connections are not visible.

42. Double-click on \Pump1.Cmd_PublicDrain. You may have to expand the name column to see the entire parameter name.

![Image of parameter list after double-click]
43. Click **OK**.

44. Highlight the parameter **Out_FillPump**. In the tag properties window, expand **Parameter Connections**.

45. Click in the box that says **‘New Connection’**.

46. Once you click in the box, click the gray browse button on the right.

47. Click the drop down arrow when the tag selection pop-up appears.

48. In the upper left corner of the parameter selection pop-up, enter **“PublicFill”**.

   NOTE: This can be a quick way to narrow down the parameters, especially if you know some key words in the tag name.

49. Double-click on **\Pump1.Cmd_PublicFill**. You may have to expand the name column to see the entire parameter name.

50. Click **OK**.
**Additional Information – Output Parameters**

**When to use Output Parameters**

Output Parameters can be connected to Controller Scoped tags, Input Parameters, and Public Parameters. This makes output parameters a great option for connecting a code module or object to an output card. It also is a good option for configuring a code module or object to invoke operations (commands, settings, etc) in other code modules or objects. Additionally, multiple connections can be configured for an output parameter. This allows one code module or object to send multiple commands to multiple modules or objects using one output parameter. This is often referred to as “fanning”.

**Output Parameter General Information**

- **Output parameters (including members) can support multiple connections.** For example, lets assume you have a BOOL input parameter in Program A and Program B named Input1a and Input1b. You are allowed to connect a single output parameter in Program C to Input1a AND Input1b. As stated earlier, this is often referred to as “fanning”.

- **Output Parameter values are refreshed AFTER each scan of a program.** Updated output parameter values will NOT be available to the parameters connected to that output parameter until the program execution is complete.

- **An Output parameter can ONLY be connected to an InOut parameter if both the Output and InOut parameters are configured as Constants.** InOut parameters will be discussed in the next section. The primary reason for this requirement is InOut parameters are passed by reference, which functions similarly to alias tags. In other words, the InOut parameter is merely a pointer to the original tag. If a non-constant InOut parameter could be connected to a non-constant Output parameter, values written to the InOut parameter would overwrite the value in the Output parameter. This is prohibited when the InOut and Output parameters are configured as Constants which means the value cannot be changed by logic. This would be a good technique for configuring a code module to monitor the output value of a different code module.
6.3: InOut Parameters

This section will cover InOut parameters. The example application contains three code modules (Pump, Tank, and Auto_Cycle). The code module interface has been defined and parameters are already created. This was done to conserve time. In this section, you will be setting up several connections to InOut parameters.

Goals:

- Create an InOut Parameter
- Configure a connection
- Understand when an InOut connection can be used

51. In the Controller Organizer, expand the Tank1 Program.

52. Double-click the Parameters and Local Tags.

Let’s assume this system has a PLC5 that is running remotely at the tank location. That PLC5 has several inputs that gather leak detection status of the tank area. This application will retrieve the leak detection status from the PLC5 using a PLC5 Word Read MSG instruction. Message instructions must reside in the Controller Tags. An InOut parameter can be used to connect directly to the Message instruction at the Controller scope. This provides an extra layer of encapsulation. The following steps in this section will walk you through the process of connecting a Message instruction to a program InOut parameter.
53. Highlight **Pv_MSG_PLC5Read**. This tag should have the Red “X”, which indicates an error.

54. Click in the box that says ‘**New Connection**’ in the tag properties window.

55. Once you click in the box, click the gray browse button on the right.

56. Click the drop down arrow when the tag selection pop-up appears.

57. Remove the checkbox on “**Show Tank1 Tags**”.

   *Note: This checkbox may be labeled as “Show Program Tags”, instead of “Show Tank1 Tags”.*

58. In the “**Show Parameters from other Program**” dropdown, select <none>. 

![Image of tag properties with highlighted checkbox and dropdown]
59. Scroll down in the list and double-click the MSG_PLC5_Read Controller Tag.

60. Click OK.

If you look in the Parameters and Local Tags for Tank1, you will also see an input parameter named Inp_LeakDetectionPLC5. Since Message instruction source and destination tags must also be at the controller scope, we created an additional parameter to connect to the result (destination) of the Message instruction.

### Additional Information – InOut Parameters

**When to use InOut Parameters**

InOut Parameters can be connected to Controller Scoped tags, Input Parameters, Public Parameters, and Output Parameters (Constants). InOut parameters are especially unique because they pass by reference rather than by value. In other words, they are merely a pointer to the original data and closely resemble the behavior of an alias tag. With that in mind, it is possible that the InOut parameter values could change during the execution of a program. Depending on your task structure, this behavior may be necessary. One example of a structure that could potentially take advantage of InOut parameters is an application that contains multiple tasks with different priority assignments. In this scenario, a lower priority task will be interrupted by a higher priority task. If the higher priority task is referencing data in the lower priority task, the InOut parameter will allow the higher priority task to point directly to a tag’s value in the lower priority task. This will ensure that the higher priority task is using the most up-to-date value of a tag. Another useful scenario where InOut parameters would be required is the use of instructions whose tags can only be placed at the controller scope. A prime example would be MSG instructions. InOut parameters would allow a program to connect directly to the MSG instruction tags in the controller scope.

**InOut Parameter General Information**

- **InOut parameters can only support ONE connection.** You cannot configure connections to any member of an InOut parameter.

- **InOut parameters are passed by REFERENCE, which means they simply point to the base tag.** In other words, when an InOut parameter is used in logic, the current value of the parameter connected to the InOut Parameter will be used.

- **An InOut parameter can ONLY be connected to an Output parameter if both the Output and InOut parameters are configured as Constants.** See the tool tip for Output Parameters for a more detailed explanation.

- **InOut parameters CANNOT be changed online, unless using the Partial Import Online (PIO).**
6.4: Public Parameters

This section will cover Public parameters. The example application contains three code modules (Pump, Tank, and Auto_Cycle). The code module interface has been defined and parameters are already created. This was done to conserve time. In this section, you will be setting up several connections to Public parameters.

**Goals:**
- Create a Public Parameter
- Configure a connection
- Understand when a Public connection can be used

61. In the **Controller Organizer**, expand the **Auto_Cycle1 Program**.

62. Double-click the **Parameters and Local Tags**

63. Click on the **Edit Tags** tab of the Parameters and Local Tags.

64. Click inside the usage box for **Sts_AutoMode**. Select the drop down arrow and select **Public Parameter**.

**NOTE:** This tag was originally configured as a Local Tag. It can easily be converted to a public parameter so other programs can get access.
65. Expand the **Parameter Connections** in the tag properties window.

The Pump program has an input parameter configured to connect to the Auto Mode status public parameter in Auto_Cycle. We will be configuring this connection in the next several steps.

66. Click in the box that says ‘**New Connection**’ in the tag properties window.

67. Once you click in the box, click the gray browse button on the right.

68. Click the drop down arrow when the tag selection pop-up appears.

69. Use the search / filter options to find \Pump1.Inp_AutoMode. Set **Show Parameters from other Programs** to **Pump1** and complete the connection.

70. Click inside the usage box for **Sts_ManualMode**. Select the drop down arrow and select **Public Parameter**.
71. Expand the **Parameter Connections** in the tag properties window.

72. Configure a connection from **Sts_ManualMode** to **Pump1.Inp_ManualMode**.

![Parameter Connections](image)

73. Navigate (scroll right) to the **Connection** column. The two connections you just created should look like the following screen shot. You may have to expand the width of the Connection column.

|          | BOOL | | Read/Write | | Decimal | |          |
|----------|------|---|------------|---|----------|---|
| Sts_AutoMode | EOSL | | | | | | Pump1.Inp_AutoMode |
| Sts_ManualMode | EOSL | | | | | | Pump1.Inp_ManualMode |

(If Connections column is not visible to the right, right-click on one of the column headers. Select **Toggle Column** and then select **Connection** if not checked.)

The remaining steps will demonstrate how to connect to members of a parameter. We will also observe how to quickly identify how many total connections are configured for a parameter.

74. Navigate to the **Parameters and Local Tags** of the **Tank1** Program.

75. Highlight **Val_LeakDetection**. Look at the **Parameter Connections** for this tag in the Properties window.

![Parameter Connections](image)

**NOTE:** Let's do a quick review of what we discussed earlier in the lab. You can quickly identify the number of connections for the parameter and its members by looking at the numbers between the brackets. The example above indicates that 0 connections are made to the parent parameter, which is **Val_LeakDetection**. However, 2 connections are made to one or more members of the parent parameter. In this example, that would mean there are two connections to one or more of the 16 bits (INT data type) of **Val_LeakDetection**. Logix Designer (version 24 and higher) software will allow you to view all of the connections to a given parameter in the parameters tab of the program properties.
Let’s create another connection to a member of the Val_LeakDetection parameter.

76. Navigate to the Parameters and Local Tags of the Pump1 Program.

77. Create a connection for Inp_PumpAreaLeak. Define this connection to \Tank1.Val_LeakDetection.0.

78. Navigate to the Parameters and Local Tags of the Tank1 Program.

79. Highlight Val_LeakDetection. Look at the Parameter Connections for this tag in the Properties window. Note how the {0:2} changed to {0:3}.

80. Next, let’s navigate to the Parameters and Local Tags of the Auto_Cycle1 Program.

81. Create a connection for Inp_LeakStatusWord. Define this connection to \Tank1.Val_LeakDetection.

82. Navigate to the Parameters and Local Tags of the Tank1 Program.

83. Highlight Val_LeakDetection. Look at the Parameter Connections for this tag in the Properties window. Note how the {0:3} changed to {1:3}.

84. The Connection column in the Parameters and Local Tags window will display the same information. You may have to configure the Connection column to be visible.

85. Mouse over the connection box for the Val_LeakDetection Public Parameter. The popup will show you all of the connections that are configured for a given parameter.
Additional Information – Public Parameters

When to use Public Parameters

Public Parameters can be connected to Input Parameters, Output Parameters, and InOut Parameters. Public parameters have the look and feel of controller scoped tags, but at the program level. The key benefit to using Public parameters over controller scoped tags is better encapsulation. If a code module or object contains data that must be globally available to other code modules, then a Public parameter would be an excellent option. Public parameters are updated as the source updates, so higher priority tasks that interrupt a program during execution will have access to any updated values in the lower priority task. Additionally, “fanning” is supported with Public Parameters. A possible scenario where this would be useful would be if a code module or object could receive a command from many modules or objects. Input parameters could be used, but an individual input parameter would be required for each connection. In other words, if a code module or object had a command that could be invoked by 10 different code modules or objects then 10 input parameters would be required. However, if the command is configured as a Public parameter, then all 10 code modules or objects could have an output connected to a single Public parameter.

Public Parameter General Information

• **Public parameters can support MULTIPLE connections.** You can configure connections to the base Public parameter or any member of a Public parameter. This includes User-Defined Structures.

• **Public parameters are updated when the source is updated.** In other words, when a Public parameter value updates, it is immediately available to any higher priority tasks that are connected to that parameter.

• **Public parameters can be aliased to Controller Scope Tags.** If this functionality is desired, it is important to remember that the alias update will be asynchronous to program execution. The public parameter will contain the real-time value of the controller scope tag.
6.5: Direct Access of Program Parameters

This section will cover the direct access of Program Parameters. You will be configuring logic and directly accessing several program parameters.

Goals:

- Configure logic to use direct access
- Understand how direct access behaves in logic

**What is Direct Access?**

Direct access allows the user to reference another program's parameters in logic without configuring parameters in the local program. For example, if Program A had an output parameter called Tank_Level, Program B could reference the Tank_Level parameter in logic without creating a corresponding parameter to connect to Program A. When direct access is implemented on Input and Public parameters, the values of the referenced parameters are updated in real time. When direct access is implemented on Output parameters, values will be updated when the program that contains the output parameter completes execution. When directly referencing Input and Public parameters, higher priority tasks will be using parameter values that are up-to-date, minimizing the risk of performing actions on stale data.

**Direct Access General Information**

- Program local tags CANNOT be accessed using direct access
- Input, Output, and Public parameters can all be directly accessed.
- Using Direct Access of input parameters is a good way to circumvent the “one connection” limit for input parameters. If you create an input but do not configure any connections for that input, you can directly reference it from multiple programs.

86. Navigate to the **MainRoutine** of the **Auto_Cycle1** Program.
87. Rung 2 should have an XIO instruction that is not tied to a tag. **Double-click** the “?” above this instruction.

![Diagram showing the Fill pump control logic](image)

We want to make sure that we turn the Fill Pump off when the Tank is full. We can use direct reference to acquire the value of the tank full status. The Auto_Cycle1 program can directly access the value of the Sts_TankFull parameter in the Tank program without creating a physical connection.

88. Click the drop-down arrow. Use the filters to find and select \Tank1.Sts_TankFull.

![Filtering and selecting Tank1.Sts_TankFull](image)

**NOTE:** The Fill Pump will now shut off when the tank is full.

89. Rung 3 should have an XIO instruction that is not tied to a tag. **Double-click** the “?” above this instruction.

![Diagram showing the Drain pump control logic](image)
90. Click the drop-down arrow. Use the filters to find and select \Tank1.Sts_TankEmpty.
   NOTE: The Drain Pump will now shut off when the tank is empty.

   ![Diagram showing logic for Drain Pump control](image)

91. **Verify** the Controller.
   
   NOTE: No Errors or Warnings should be present. If the Error window is not visible, click Alt+1.

   Let's download the program to the controller and see how the program operates.

92. Click on **Communications => Who Active** at the top of the screen.
   
   NOTE: RSLinx may take a few moments to open.

   ![RSLinx screen showing Who Active](image)

93. Expand the Ethernet/IP Driver **AB_ETHIP-1**

   ![Expanded Ethernet/IP Driver](image)
94. Select **192.168..1.218, 1756-L85E**  You do not need to expand the controller.  Press **Download**.

95. When prompted, press **Download**.

96. When prompted to change the controller back to **Remote Run**, Click **Yes**.

97. Make sure the switch at **DI4** is turned to the **RIGHT**.  This switch allows the user to select between Auto (Right) and Manual (Left) Mode.

98. When the green light at **DO0** is illuminated, the Fill Pump is ON.

99. When the green light at **DO2** is illuminated, the Drain Pump is ON.

100. The Voltmeter at **AO0** simulates the tank level.  0 volts indicates EMPTY, 10 volts indicates FULL.

101. Turn the **AI0** Knob to **5**.  The potentiometer knob simulates the pump speed.  As you turn the knob clockwise, the pump speed will increase.

    **NOTE:**  You should notice the tank slowly filling and draining.  As the voltmeter swings from left to right, the DO0 green light should be illuminated (filling).  As the voltmeter swings from right to left, the DO2 green light should be illuminated (draining).

102. Turn the **AI0** Knob to **10**.

    **NOTE:**  The rate at which the tank is filling and draining should have increased.

103. Turn the **DI4** switch to the **LEFT** position.  This will put the system into Manual Mode.

    **NOTE:**  The Green lights should both turn off.  The voltmeter should stop moving.  Since the system is now in manual mode, you must manually operate the fill and drain pumps.

104. **Press and HOLD** the green button at **DI0**.  This will manually turn the fill pump ON.

    **NOTE:**  You should notice the green light will turn ON and the voltmeter will being to increase.  When the tank is full, the green light should turn off even if the button remains depressed.

105. **Press and HOLD** the green button at **DI2**.  This will manually turn the drain pump ON.

106. **Go Offline** with the controller.

107. Close Logix Designer.  If prompted to save the file respond with **No**.

---

**Congratulations!**

You have just finished this Lab

[Click to Proceed to Contents](#)
Lab 7: Add-On Instructions (AOI) <25 minutes>

About This Lab

Version 16 of RSLinx5000 Logix introduced the concept of reusable code objects called Add-On Instructions (AOI). Add-On Instructions allow you to encapsulate your most commonly used logic as sets of reusable instructions, similar to the built-in instructions already in the Logix controllers. This saves you time by allowing you to easily re-use sets of your commonly used instructions in your projects and promotes project consistency because commonly used algorithms will all work in the same manner, regardless of who is implementing the project.

Add-On Instructions may be created using the standard ladder, function block diagram, and structured text editors already available in Studio 5000® Logix Designer. Once created, an Add-On Instruction may then be used in any of the Studio 5000® Logix Designer editors without any additional effort on your part.

Online troubleshooting of your Add-On Instructions is simplified by the provision of context views which allow you to visualize the logic for your instruction for a specific instance of its use. Custom help for your instructions makes it easier for users to have a successful application of the Add-On Instruction.

Finally, you can use the Add-On Instruction’s source protection capability to prevent unwanted changes to your instruction and/or to protect your intellectual property.

In this lab we will take all of the conveyor code we were using in the previous lab and encapsulate it within a single Add-On instruction. We will then use the Add-On instruction in another project to show how much time you can save with them. In this lab you will:

- Get an overview of AOI Basics
- Create an AOI
- Use an AOI
- Apply signature to an AOI
- Export / Import AOI with a signature
7.1 AOI Basics
This section provides an overview of the basic parts of an Add-On Instruction and their uses.

1. General Information
The General tab contains the information from when you first created the instruction. You can use this tab to update the information. The description, revision, revision note, and vendor information is copied into the custom help for the instruction. The revision is not automatically managed by the software. You are responsible for defining how it is used and when it is updated.

2. Parameters
The Parameters define the instruction interface; how the instruction appears when used. The Parameter order defines the order that the Parameters appear on the instruction call.

NOTE: InOut tag values are not displayed even when Required and Visible is selected because they can be configured as arrays - displaying arrays can consume excessive amounts of space.
3. Local Tags
Local Tags are hidden members private to the instruction. Local Tags are not visible outside the instruction and cannot be referenced by other programs or routines.

4. Data Type
Parameters and Local Tags are used to define the data type that is used when executing the instruction. The software builds the associated data type. The software orders the members of the data type that correspond to the Parameters in the order that the Parameters are defined. Local Tags are added as hidden members.

5. Logic Routine
The Logic routine of the Add-On Instruction defines the primary functionality of the instruction. It is the code that executes whenever the instruction is called. Shown below is the interface of an Add-On Instruction and its primary logic routine that defines what the instruction does.
6. Optional Scan Mode Routines

![Add-On Instruction Definition - P_Motor v2.0 00 Release](image)

The controller prescans and postscreans the AddOn Instruction Logic routine but will not execute the Logic routine when Enable is false. Optional Prescan, Postscan and EnableIfFalse routines may be configured below.

- **Prescan routine:** Executes prior to first scan on transition from program to run mode.
  - Execute Prescan routine after the Logic routine is prescanned
  - **Delete**
  - **Go To**

- **Postscan routine:** Executes on last scan of step if SFC is configured for automatic reset
  - Execute Postscan routine after the Logic routine is postscanned
  - **New**
  - **Go To**

- **EnableIfFalse routine:** Executes when the Enable parameter is false
  - Execute EnableIfFalse routine
  - **Delete**
  - **Go To**

![Add-On Instruction Definition - P_Motor v2.0 00 Release](image)

7. Change History

The Change History tab displays the creation and latest edit information that is tracked by the software. The **By** fields show who made the change based on the Windows user name at the time of the change.

![Add-On Instruction Definition - P_Motor v2.0 00 Release](image)
8. Help

The Name, Revision, Description, and Parameter definitions are used to automatically build the Instruction help. Use the Extended Description Text to provide additional Help documentation for the Add-On Instruction. The Instruction Help Preview shows how your instruction will appear in the various languages, based on Parameters defined as Required or Visible.
7.2 Creating an Add-On Instruction (AOI)

In this section of the lab we will take one section of the conveyor code we worked with in the last lab and use it to create part of our AOI.

In this section of the lab, you will open an existing program and create the AOI.

9. Open CLX_S7_00.ACD using in the C:\Lab Files\Advanced Logix\Lab 7\AddOn Instructions AOI folder.

10. From the Controller Organizer, expand the Assets folder.

   Currently each conveyor code section is made up of one program and four routines. By using an AOI we can reduce all of this to one custom instruction that can be used anywhere in the application.

11. Right click on the Add-On Instructions folder and select New Add-On Instruction… from the dropdown menu.
12. Complete the ‘New Add-On Instruction’ dialog as follows:

- **Name:** Conveyor_Core
- **Description:** This code contains all the core conveyor logic
- **Type:** Ladder Diagram
- **Revision:** 1.0
- **Vendor:** Rockwell Automation

13. Click **OK** button to accept.
14. The instruction definition will open.

15. Click on the **Parameters** tab to view the parameters for the instruction.
'EnableIn' and 'EnableOut' parameters are predefined and are added by default to each Add-On Instruction. The 'EnableIn' is manipulated by factors unique to each language environment and can be used for specialized functionality. This is an advanced topic and is beyond the scope of this lab. The 'EnableOut' generally follows the state of the 'EnableIn' but can be manipulated by user programming.

The Parameters tab is a tag database editor window for parameters specific to this Add-On Instruction with the inclusion of definition specific fields:

- **Usage** allows the user to designate the parameter as:
  - **Input** An input to the instruction (atomic type)
  - **Output** An output from the instruction (atomic type)
  - **InOut** A parameter which is passed 'by reference' to the instruction (any data type including UDTs, arrays, etc)

- **Default** allows the user to specify a default value for the associated parameter. This is the value the parameter assumes when the instance tag for an instruction call is first created.

- **Req** - A parameter which has been checked as 'Required' will force the user to enter a tag or make a FB (Function Block) connection to this parameter when an instruction instance is used in a routine. This modifier defines verification behavior in the languages as will be seen later. A 'Required' parameter is also 'Visible' by default.

- **Vis** - A parameter which has been checked as 'Visible' will be visible by default on the instruction when it is used in a routine. Use of this modifier causes differing results depending upon the data type and the language (LD, FB, ST, SFC) in which an instruction instance is used.

- **Description** field is important because any information used here will be ‘passed through’ to any instruction instance of the user program. This ‘self-documents’ the instruction for usage, units, etc.

We will be copying and pasting code from an existing routine for this lab but we need to create some new tags first.

16. Enter the following tags as shown on the Parameter tab.

   Note: Parameters are **Input, Output, and InOut** tags only. Local tags are entered on a separate tab. Click the **Apply** button to accept changes.
17. Click the *Local* Tags tab and enter the tags as shown.

18. Click *Apply*

19. Click on *OK* to close the dialog.
7.3 Entering the Logic

We are going to re-use and copy the logic we have to save lab time. Typically a user would enter logic from scratch.

20. From the Controller organizer locate the **C_Start_Stop** routine and double-click it to open it.

![Controller organizer with C_Start_Stop routine](image)

21. From the toolbar menu select **Edit > Select All**. This will select all the rungs in the routine.

22. Right click on **Rung 0** and select **Copy Rung**.

23. Expand the **Add-On Instructions** folder if it is not already expanded in the Controller organizer.

24. Double-click on **Logic** to open it.

25. Right click on the End rung and select **Paste**.

26. Delete the empty rung that appears at the top.

27. All of the logic should now appear without any errors.

   Note: Errors may occur if the tags were not typed in correctly in the definition of the AOI. A complete working example is provided in the next step so don’t worry if errors exist at this point.
That's it. We just created our first Add-On Instruction. In real life we probably would want to put all of the routines in it but we just wanted to show how to construct one.

In the next section we will open an existing program and begin working with a complete AOI.

Close the file within Logix Designer using **File > Close**. Respond with **No** if prompted to save the file.
7.4 Using the AOI in RLL (Ladder Logic)
Add-On Instructions may be created using the standard Ladder, function block diagram, and structured text editors already available in Studio 5000® Logix Designer. Once created, an Add-On instruction may then be used in any of the Studio 5000® Logix Designer editors without any additional effort on your part. In this section of the lab, you will open an existing program and create the AOI.

28. Open **CLX_S8_00.ACD** using the File > Open menu browsing to **C:\Lab Files\Advanced Logix\Lab 7 AddOn Instructions AOI** folder.

   This file has the completed **Conveyor_Core** AOI already done for you. All three Routines that make up one conveyor have been combined into one custom instruction.

29. Under AOIs, expand the **Conveyor_Core** AOI until you can see the Parameters and Local Tags.

30. Right click on **Conveyor_Core** and select **Open_Definition**.

   This will open the definition so you can see all the tags that are now part of the instructions definition.

31. Click the **Parameters** tab.

32. Notice which ones are marked **REQ** and **VIS**. This will determine how the instruction looks and acts once we place it in an application.
33. Click on the **Logic** button in the bottom left of the dialog to open the AOI logic.

34. You should see ten rungs total in the AOI.
7.5 Using the AOI

35. In the MainTask, Open the Conveyor_1 routine located in the Conveyor program.

36. You should see a blank rung that we will can add the AOI to.

37. Select the **Add-On** tab and then click the **Conveyor** button.

38. This will add a **Conveyor** instruction to the rung.
AOI tag structure: At this point the instruction behaves like any other Logix Designer instruction. It has inputs and outputs that need to be defined and other parameters that need to be filled in – just like a Timer does.

The parameters with a single question mark (?) after them require a tag name to be entered. The parameters with a double question mark (??) after them are tag value fields. These fields show the current values of those parameters.

39. Right click on the first question mark and select **New Tag**.

40. Complete the **New Tag** dialog as shown.
Note: Data Type is *Conveyor_Core*. This is what we named our AOI. This follows the same conventions as any other built in instruction in Logix. If this were a TON instruction, the Data Type would be Timer.

41. Click **Create** to accept.

42. The rest of the arguments are inputs. We can assign some real I/O to these inputs the way we would for any other input arguments.

43. Double-click on the *Motor_Overload_Input* question mark and select **Local:3.I.Pt[0].Data** as shown:

44. Do the same for the rest of the AOI inputs, selecting the next sequential input points for each input in the instruction, until all of the question marks are gone.

45. The rung should match the image below:

46. Verify the routine by clicking on the Verify icon.
7.6 Creating Alias Tags

47. Click on the Motor_Overload_Input address and type **My_Motor_Overload**. After you accept it you should see question marks for the value because the entered tag name does not exist yet.

48. Right click on the new tag name and select **New “My_Motor_Overload”**
49. Fill in the dialog as shown. Make sure you fill in the Alias part as shown by using the pull down menu and selecting the correct I/O point.

![New Parameter or Tag dialog](image)

50. Click **Create** to accept it. The instruction should now have your Alias filled in.

![Parameter connection](image)
7.7 Using an Add-On Instruction Output in Ladder

51. On the next rung below the AOI, create the logic pictured below to demonstrate how to access non-visible tags from an AOI programmatically.

How do you know what the internal elements are named? The AOI follows the same conventions as any other instruction does.
7.8 Using the AOI in a Function Block (FB) Routine

52. Create a new routine named *Conveyor_2* by right clicking on the *Conveyor* Program and then selecting **Add** > **New Routine**.

53. Fill out the dialog as shown specifying **Function Block Diagram** as the Type and click **OK** button when you are finished.

54. Double-click on the new **Conveyor_2** routine to open it.
55. Select the **Add-On** tab and click the **Conveyor** button. The **Conveyor** instruction will be added to the page.

Note, the pins associated with parameters designated as **Visible** in the instruction definition are visible on the instruction by default.

The data tag for the instruction **Conveyor_Core_01** has already been created. This conforms to the behavior of any instruction created in the Function block environment.

The nubs on the input pins is an indication that this parameter requires a connection. The instruction will not successfully verify until all “**Required**” parameters have connections.
Let’s add 6 input references for each of the 6 input pins on the AOI.

56. Click the Input Reference (IREF) button six times as the AOI requires six inputs.

57. Drag the IREF’s to align roughly with the AOI inputs on the sheet and wire them to the Motor_Overload_Input connectors.

58. Select the addresses for all the IREF’s as shown below for the required input points. You could use an Alias here just like we did for the ladder instruction.

59. The instruction will now verify. Notice the “x” is gone from the upper left corner.
7.9 Re-Using an AOI

Remember in the PIO lab how we saved time by copying and pasting or exporting and importing code? We can do the same things with AOI’s to save even more time or to simplify program design and re-use. In this section we will also apply security to a routine and an AOI.

**Storing your Instructions**

There are two ways to store and use a group of instruction definitions together.

- Keep them in a project file (.ACD file) and then use Copy/Paste feature to use them
- Export them to an .L5X file and then import them as you need them

**Copy and Paste**

60. Continue working with *CLX_S8_00.ACD* file. (should be the file that you currently have open)

61. From the controller organizer in our Conveyor application, right click on the *Conveyor_Core* AOI and select *Copy*.

62. Open a second instance of Studio 5000® Logix Designer using the shortcut for Studio 5000® Logix Designer on the desktop or the button selecting Programs\Rockwell Software\Studio 5000

63. Begin a new file by selecting *New Project* from the Quick Start page.
64. **Select a Logix Controller.** For this example we selected a 1756-L85E located under ControlLogix 5580 Controller family. **Enter the name AOI_LAB**

![Select Logix Controller](image1)

65. **Click Next** when finished.

66. Be sure to select **Revision 31** for Logix Designer.

![Revision Selection](image2)
67. **Click Finish** once complete.

   Note: It will take a minute to create the new ACD file.

68. In the Controller organizer right click on **Add-On Instructions**

69. Select **Paste With Configuration**.

   ![Image of Controller Organizer with Add-On Instructions selected for Paste With Configuration](image)

   Note: If you select Paste the copy may fail. “Paste With Configuration” opens the Partial Import Online (PIO) dialog window which is different than just a Paste.

70. Click **OK**.
71. The *Conveyor_Core* AOI should now be present and ready for our use in the new program.

72. Close this instance of Logix Designer. Respond with *No* if prompted to save the file.
7.10 AOI Signatures

Copying and pasting requires you to have two instances of Studio 5000® Logix Designer open at the same time. This is fine if you are using your own AOI's. How do you send someone else a copy of the AOI? What if you want to apply security to the AOI? Perhaps you want the other user has edited the AOI?

AOI Signatures allows the application developer to monitor changes to an AOI. The AOI signature value can be monitored to record “version” of AOI that was created. Users can view AOIs with Signatures, but they will need to remove the Signature to edit the AOI. Signatures will also need to be removed and reapplied when the Major revision of a project has changed. 

In this section of the lab we will:
- Apply a Signature to the Conveyor_Core AOI
- Export the AOI
- Import the AOI in a new application

Applying Signatures

Before we export the Conveyor_Core AOI we will apply a signature to it. Then we will import it into a new ACD file.

73. Continue using the CLX_S8_00.ACD file.
74. In the controller organizer, navigate to the **Conveyor_Core** AOI. Double-click on the AOI to display its properties. Navigate to the **Signature** tab, shown below.

![Signature Tab](image1.png)

75. Click on **Generate**, to generate a new Signature for this AOI.

![Generate Button](image2.png)
76. A warning message will be displayed. **Click Yes** to confirm and generate the instruction signature.

![Warning Message]

**Note:** AOIs with Signatures cannot be modified. The **AOI signature will need to be removed before the AOI can be modified.** There is not any security on removing AOI signatures, so anyone can remove the Signature and change the AOI. But, **if a new AOI signature is created it will not match the original signature.**

77. Below is the AOI with the Signature.

**NOTE:** Your Signature value will be different than what is shown below.

![AOI with Signature]

78. **Click OK.**
In the Controller Organizer, the icon for the Conveyor_Core AOI has now changed since a Signature has been added. The blue dot in the icon indicates that the AOI has a signature.
7.11 Export AOI
We will take a look at how to export an AOI and then import it into a new Logix Designer application.

80. In the controller organizer, right click the **Conveyor_Core** AOI and select **Export Add-On Instruction**.
81. Enter **Converor_Core.L5X** as File Name for the AOI and then click **Export**.  
Note: Remember the folder name that you exported the AOI to.

Note: The *Include all referenced Add-On Instructions and User-Defined Types* checkbox in the **Export AOI** dialog box that is checked by default.

Next we will create a new Logix Designer Application

82. Open a second instance of Studio 5000® Logix Designer using the shortcut for Studio 5000® Logix Designer on the desktop or the button selecting Programs\Rockwell Software\Studio 5000.
83. Begin a new file by selecting **New Project** from the Quick Start page.

84. Select a Logix Controller. For this example we selected a 1756-L85E. Enter the name **AOI_LAB2**.

85. Click **Next**.

86. Select Revision **31** of Logix Designer and click **Finish**.
87. In the controller organizer, right click on the Add-On Instructions folder and select Import Add-On Instruction.

88. Browse to Conveyor_Core.L5X file that you created above and click Open.

89. This will bring up the Partial Import Online (PIO) dialog box. Our operation is to create a new AOI in this new application.

90. Click OK to import the AOI.

91. Now you can view the imported AOI in the new Logix Designer application.
92. You can also compare the AOI Signature values in each of Logix Designer application. They will match.

93. Close the both instances of Logix Designer using **File > Close**. Respond with **No** if prompted to save the file.
Lab Summary

This lab shows that with Add-On Instructions you can generate code in hours versus days. When you use code segments over and over again you can simply place an AOI in your program instead of re-generating the code again and again. We saw that we can develop complex code segments and ship them to customers that need. We can apply signatures to AOIs that will allow customers to view and change the code inside, but give use indication that something has changed.

In the first few labs we have moved from all of our logic bunched in one big program to a single instruction that is very compact and easy to use.

Congratulations!

You have just finished this Lab

Click to Proceed to Contents
Lab 8: Using a SD card with a Logix Controller  <10 minutes>

About This Lab
All Logix Controllers have a SD card slot designed to read and write data to a SD card. This allows critical process data or recipes to be saved external to the memory of the controller. Custom message instructions are used to do file operations on the SD card. The messages are used to create, write, read, and delete both files and directories on the SD card.

Before you Begin
This lab requires that a SD card be installed in the controller. All ControlLogix and CompactLogix controller ship with SD Cards installed. Please double check that your controller contains a SD card. It may have been misplaced in a prior lab session.

Using a SD card reader connected to the PC copy files from C:\Lab Files\Advanced Logix\Lab 8 Using a SD card with a Logix Controller\Files to be placed on SD onto your SD card. The SD card slot is covered by a door on the front of the controller.

Note: Get a pre-loaded SD card from one of your lab instructors before beginning the lab.

8.1 Opening an Existing Controller Project
In this section of the lab, we load the controller project.

1. Double-Click on the Lab Files folder on the desktop.

2. Navigate to C:\Lab Files\Advanced Logix\Lab 8 Using a SD card with a Logix Controller double-click the project named CLX_S13_00.ACD to open it in Logix Designer.

   NOTE: This may take up to a minute to open.

3. Click on Communications => Who Active at the top of the screen.
   NOTE: RSLinx may take a few moments to open.
4. Expand the Ethernet/IP Driver **AB_ETHIP-1**

5. Select **192.168.1.218, 1769-L85E**. You do not need to expand the controller. Press **Download**.

6. When prompted, press **Download**.

7. If prompted to change the controller back to **Remote Run**, Click **Yes**.

8. If not prompted, changed the controller mode to **Remote Run** in Studio 5000 Logix Designer.
8.2 Using code to list files and directories

The code used in this section messages at the lowest level to perform operations on the SD card. The interface will be numerical commands to the add-on-instructions and routines developed to interface to the SD card. You will enter numeric commands and see results displayed in the tags of the controller.

<table>
<thead>
<tr>
<th>Functions to Write and Read SD card</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – Clear data from UDT</td>
</tr>
<tr>
<td>2 – Read data from file with offset</td>
</tr>
<tr>
<td>3 – Write data to file with offset</td>
</tr>
<tr>
<td>4 – Append Data to file</td>
</tr>
<tr>
<td>5 – Delete file/directory</td>
</tr>
<tr>
<td>6 – Rename file/directory</td>
</tr>
<tr>
<td>7 – Create directory</td>
</tr>
<tr>
<td>8 – List files/directories with search pattern</td>
</tr>
<tr>
<td>9 – List file attributes</td>
</tr>
<tr>
<td>10 – List directories only</td>
</tr>
</tbody>
</table>

Note: SD card Logix functions are the same as the CompactFlash function used in prior Logix controllers.

Note: All of the current Logix controller use SD cards. An SD card comes shipped with every controller.

9. Double-click to open **Controller Tags** in the project tree.
10. Expand the arrow symbol next to the tag **CF_RW_functions** and expand the arrow symbol next to **CF_RW_functions.strings** array member. Make sure you are in **Monitor Tags** and not **Edit Tags**.

11. In **CF_RW_functions.command** member, set it to a value of 8. After the number is entered, Press **Enter** on the keyboard. This completes the write to the controller tag. It will disappear immediately.

12. Look at the value of **CF_RW_Functions.result**. When positive, it displays the number of files found. When negative an error has occurred performing an operation.

Note: You may have additional files on your SD card.

**If you have an issue that results in a negative error value please alert the instructor.**
13. Names of files and directories found on the SD card are displayed in the 
*CF_RW_Functions.string* array members.

Note: You may have different or additional files on your SD card.

The 8 function is a list command. Equivalent to a DOS ‘dir’ or Unix ‘s’ command. It lists all files and directories. Using the ‘name’ member you can search for specific file or directories. Simply set the tag ‘name’ to ‘o’ and it will display all files with an ‘o’ in the name.

14. Enter a 1 in the command member. It will disappear and so will result and string values.

The 1 function is a clear command. It clears all values for arrays used to message the controller. It does not send any messages to perform SD card operations.
15. Enter a 9 in the command member. It will disappear, and in the 'string' values, we see the list of files with attributes listed and sizes in Kbytes.

Note: You may have different or additional files on your SD card.

| CF_RW_functions.strings | \{'Logic': '---D-- 16 KB', 'RockwellAutomation': '---D-- 1...', 'Recipe 1': '-----N 0 KB', 'NewDirectory': '---D-- 16 KB', 'fiveK.txt': '-----N 4 KB', 'ONE.TXT': '-----N 0 KB', 'LOGDIR': '---D-- 16 KB', 'morefiles': '---D-- 16 KB', 'testfiles': '---D-- 16 KB', 'Archive.txt': '-----N 0 KB', 'Skexppublickey.txt': '-----N 0 KB', 'OLDLOG.TXT': '-----N 2 KB' |

The '9' function is useful for viewing files and their attributes including size.
8.2 Create a Directory

16. Enter a 10 in the command member. It will disappear, and in the ‘string’ values, we see the list of directories on the SD card.

<table>
<thead>
<tr>
<th>CF_RW_functions.strings</th>
<th>{...}</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF_RW_functions.strings[0]</td>
<td>'Logix'</td>
</tr>
<tr>
<td>CF_RW_functions.strings[1]</td>
<td>'RockwellAutomation'</td>
</tr>
<tr>
<td>CF_RW_functions.strings[3]</td>
<td>'LOGDIR'</td>
</tr>
<tr>
<td>CF_RW_functions.strings[4]</td>
<td>'morefiles'</td>
</tr>
<tr>
<td>CF_RW_functions.strings[5]</td>
<td>'testfiles'</td>
</tr>
</tbody>
</table>

The ‘10’ function is used to list directories only.

17. Enter the text NewDirectory in the CF_RW_functions.name member. (Click the browse button on the left side of the field to bring up the string editor).

18. Set the Command value to 7. You have now created a directory called ‘NewDirectory’. To verify and see this result, set the Command to 1 and then set Command to 10.

<table>
<thead>
<tr>
<th>CF_RW_functions.strings</th>
<th>{...}</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF_RW_functions.strings[0]</td>
<td>'Logix'</td>
</tr>
<tr>
<td>CF_RW_functions.strings[1]</td>
<td>'RockwellAutomation'</td>
</tr>
<tr>
<td>CF_RW_functions.strings[3]</td>
<td>'LOGDIR'</td>
</tr>
<tr>
<td>CF_RW_functions.strings[4]</td>
<td>'morefiles'</td>
</tr>
<tr>
<td>CF_RW_functions.strings[5]</td>
<td>'testfiles'</td>
</tr>
</tbody>
</table>

The ‘7’ function creates a directory on the SD card.
8.3 Create, Edit, and Delete a File

19. Set the **Name** member to **mistake.tx**. Set the **REQ_Length** to **235**.

<table>
<thead>
<tr>
<th>CF_RW_functions</th>
<th>(...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF_RW_functions.command</td>
<td>0</td>
</tr>
<tr>
<td>CF_RW_functions.index</td>
<td>0</td>
</tr>
<tr>
<td>CF_RW_functions.offset</td>
<td>0</td>
</tr>
<tr>
<td>CF_RW_functions.REQ_length</td>
<td>235</td>
</tr>
<tr>
<td>CF_RW_functions.DN_length</td>
<td>32768</td>
</tr>
<tr>
<td>CF_RW_functions.result</td>
<td>5</td>
</tr>
<tr>
<td>CF_RW_functions.name</td>
<td>'mistake.tx'</td>
</tr>
<tr>
<td>CF_RW_functions.newname</td>
<td></td>
</tr>
</tbody>
</table>

20. Set Command to **3**.

The ‘3’ function creates a file on the SD card.

21. You have now created a file called **mistake.tx**.

22. To verify this result set Command to **1** then to **8**. You will see **mistake.tx** in list of files on the SD card.

<table>
<thead>
<tr>
<th>CF_RW_functions.strings</th>
<th>(...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF_RW_functions.strings[0]</td>
<td>'Logix'</td>
</tr>
<tr>
<td>CF_RW_functions.strings[1]</td>
<td>'RockwellAutomation'</td>
</tr>
<tr>
<td>CF_RW_functions.strings[2]</td>
<td>'Recipe1'</td>
</tr>
<tr>
<td>CF_RW_functions.strings[5]</td>
<td>'fiveK.txt'</td>
</tr>
</tbody>
</table>

23. Now we will rename this file since we have a mistake in the file extension. Set **name** member to **mistake.tx**. Set the **newname** member to **correctedmistake.txt**.

<table>
<thead>
<tr>
<th>CF_RW_functions.result</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF_RW_functions.name</td>
<td>'mistake.tx'</td>
</tr>
<tr>
<td>CF_RW_functions.newname</td>
<td>'correctedmistake.txt'</td>
</tr>
</tbody>
</table>

24. Set Command to **6**.

The ‘6’ function renames a file on the SD card.
25. Check the result and see that the name was changed. Set Command to 1 then to 8. The name has changed in the list.

| CF_RW_functions.strings[4] | 'correctedmistake.txt' |

26. To delete the file, Set Name member to **correctedmistake.txt** and Set the Command to 5.

The '5' function deletes a file on the SD card.

27. Check the result and see that the file was deleted from the SD card. Set Command to 1 then to 8.

Note: You may have additional files on your SD card.
8.4 Read Data from SD card

Now you will read data from the SD card and compare to how the file looks on a PC.

28. Open *Fivek.txt* in notepad by browsing to `C:\Lab Files\Advanced Logix\Lab 8 Using a SD card with a Logix Controller\Files to be placed on SD folder in windows explorer`. It is simply a repeating pattern of alphabetic characters.

29. Revert back to Studio 5000 Logix Designer and the controller tags window.

30. Set *Name* member to *fiveK.txt*. Set *REQ_Length* member to 200 and the Command to *2*.

31. Expand the `CF_RW_functions.data` array and set Style to *ASCII*. Compare this to the data we see in Notepad.

The ‘2’ function reads a file on the SD card.
32. Set the Offset member to 990 and then set Command to 2.

33. Array elements 7-9 are now ‘X’, ‘$r’, and ‘$I’. In Notepad, this corresponds to the end of the tenth line, where control characters represent Carriage Return and Line Feed. The offset is the number of bytes from the start of the file, where the code starts to read. Since all the lines in the text file are exactly 100 characters, that means just before the end of the tenth line.
8.5 Storing and Loading Recipe Data

34. In Studio 5000 Logix Designer, expand the **Check_CFcode** program and double-click to monitor the **Parameters and Local Tags**.

35. Expand the **active_recipe** tag. You can leave the names or values provided or change them as you desire.

Note: Names and values where already populated. You can change them to verify that they are loaded on the SD card in a later step.
36. Open the `recipe_handling` routine and trigger rung 0 (toggle the bit `store_recipe2recipe1`). It will immediately reset back to 0.

37. Switch back to `Controller Tags` (CTRL-Tab) or double-click `Controller Tags` again to see that `DN_Length` and `Result` are both equal to 216.

38. Return to `Check_CFcode` and double-click on `Parameters and Local Tags` again. Change some names and values in the `active_recipe` tag.

39. Open the `recipe_handling` routine and trigger rung 2 (toggle the bit `load_recipefromrecipe1`).
40. Check the values by double-clicking on Parameters and Local Tags. The values were read back from the recipe 1 file.

41. Open the Recipe_Handling routine and trigger rung 1 (toggle the bit store_recipe2recipe2).

42. Switch back to our Controller Tags window. Set Command to 1 then to 8.

43. Two extra files called recipe1 and recipe 2 have been created on the SD card.

44. If a SD card reader is available on you lab PC you can view the files. Please remove the controller and carefully place the SD card into the reader. Please feel free to view the files you created and how the data appears.
Lab Summary

Use of an industrial SD card is recommended (part number 1784-SD1; 1GB Capacity) for reading and writing data. One must make sure the card is formatted correctly to accept file reads and writes. This open interface to read and write data allows the user a number ways in which to use this feature.

**Warning:** The life expectancy of flash media is strongly dependent on the number of write cycles that are performed. Even though flash media controllers use wear leveling, users should prevent frequent writes. This is particularly important when logging data. The best approach is to log data to a buffer and write that buffer data to the SD card.

**Congratulations!**

You have just finished this Lab

[Click to Proceed to Contents]
Lab 9: Controller Change Log  <10 minutes>

About This Lab

The Controller Change Logging feature allows a developer to implement controller level change management. A probable scenario for implementation is for a machine developer, such as an OEM, who would like to understand what controller changes occurred with a machine that has been put into service for a customer. To view the diagnostic information in the log file a user removes the SD card then uses a SD card reader to view a log file in TSV (Tab Separated Variable) format on a PC.

The changes are logged to internal memory and include:

- Download
- Online edits occurred
- Forces enabled/disabled
- Mode changes (Program, Run)
- Major faults.

The Logix Controller will temporarily store up to 100 log entries in its internal memory which is separate from the user (Studio 5000 Logix Designer project) memory. When the Log file is 80% full in the controller it can automatically write to a file on a SD card or a user can force a write to the SD card using a message instruction. In the event of a power loss (without battery backup) or a controller firmware update, the log files in the controller memory are erased.

In this section, you will view sample code that is provided as a resource with the Studio 5000 Logix Designer to develop an understanding of how change management is configured.

Before you Begin

This lab does not require that a SD card be installed in the controller. But, one maybe used to view the results of this lab procedure. Availability of cards maybe limited please contact your instructor if you wish to complete this section of the lab.

Note: Your controller has an SD in the front behind a door.

In this section of the lab, we load the controller project.

1. Double-Click on the Lab Files folder on the desktop.

2. Navigate to C:\Lab Files\Advanced Logix\Lab 9 Controller Change Log and double-click the project named CLX_S14_00.ACD to open it in Logix Designer.

You can find the project in this directory:

NOTE: This may take up to a minute to open.
3. Click on **Communications** => **Who Active** at the top of the screen.  
   **NOTE:** RSLinx may take a few moments to open.

4. Expand the Ethernet/IP Driver **AB_ETHIP-1**

5. Select **192.168.1.218, 1756-L85E**. You do not need to expand the controller. Press **Download**.

6. When prompted, press **Download**.

7. If prompted to change the controller back to **Remote Run**, Click **Yes**.

8. If not prompted, changed the controller mode to **Remote Run** in Studio 5000 Logix Designer.
9. Expand the Tasks folder in the controller organizer the project tree to reveal the following MainProgram and its associated routines.

- MainRoutine: initial routine that calls other routines in the program.
- Commands: Write, auto-write setup, special counter configuration, and custom entry
- Counters: contains all the log counter information
- CustomEntryFormatting: optional routine for demonstrating a custom entry.

10. Open the MainRoutine


12. Toggle the START bit by right clicking the XIC and selecting Toggle Bit or select the XIC and press Ctrl-T on Rung 1. This starts the initialization routine preconfigured in the code that configures message instructions for the proper slot number of the controller. This will ensure that all of the message instructions will have the correct path.
13. Double-click to open the **Counters** routine. Each rung has a different Get System Value or Set System value associated with the **Controller logging**. Each of the **GSV** instructions works as described below.

<table>
<thead>
<tr>
<th>GSV Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ControllerLogTotalEntryCount</td>
<td>Number of entries added to the log since the last time the value was reset.</td>
</tr>
<tr>
<td>ControllerLogUnsavedEntryCount</td>
<td>Number of entries in the controller RAM not yet written to the SD card.</td>
</tr>
<tr>
<td>ControllerLogExecutionModificationCount</td>
<td>Number of entries that are related to modifications to the behavior of a running controller such as program/task properties change, online edit, controller timeslice modification, and I/O &amp; SFC forces (optional).</td>
</tr>
</tbody>
</table>
Rungs 4 and 5 allow us to programmatically set or reset the Modify execution Count and Total Entry Count values via \textit{Set System Value (SSV)} instructions.

14. Turn the controller key switch on the demo box to \textit{PROG} (Program) mode. Now turn the key switch all the way to \textit{RUN} and then to \textit{REM} (Remote), putting the controller in Remote Run mode. The \texttt{ControllerLogTotalEntryCount} in Rung 1 is incremented by 2, because the change to \textit{Program} mode is logged and then change to \textit{Run} mode is logged.

15. Leave the Controller in \textit{Run} mode and in the \texttt{MainRoutine}, toggle the \texttt{START} bit in Rung 1 again.

16. Double-click to open the \texttt{Commands} routine. It contains a series of message instructions to configure when and how often the controller writes to the SD card. Additionally, a user may choose to add/remove the logging of program forces and also the ability to create a custom log message. Rung 1, when toggled true, will write from internal controller memory to the SD card. This is currently done on command by toggling the Write bit.

Note: You will have to toggle each of the bits in the following steps to enable each Controller Log function.

Note: The write message will error out if No SD card exists in the controller.
17. Rung 2 is a message that can be used to turn on and off the automatic write to a SD card. Rung 3 is a GSV that reads that status back. By setting the Controller to Automatically Write Entries to the SD card, at 80% of the controller’s internal memory the log entries will be automatically be copied to the SD card.

Using the MSG instruction to read back the results of a MSG to set a value is good programming practice to see that your configuration message was received.

Rung 4, 5, and 6 are to configure whether you log I/O forces or Sequential Function Chart forces that take place in the program. Rung 4 is the configuration bit which is messaged to the controller. Rung 5 is the message instruction that sends that configuration bit to the Controller. Rung 6 is a MSG that reads that status information back.

18. Rungs 7 and 8 are to configure a custom log entry. In addition to the default types of changes that will be logged automatically, a user is able to configure custom messages to be logged. In this case, Rung 7 triggers the **CustomEntryFormatting** subroutine which populates a tolerance value into a string. Rung 8 is used to message the formatted string structure to the controller to be placed into the log file.
19. **Custom Entry Logging** will allow a user to setup a number of different entries to be triggered in the user program. Parameter changes, I/O state changes, or custom process related alerts are examples of log entries that could be useful to a programmer to troubleshoot or track important information for a system. The log entries are limited to the ideas of the programmer.

20. Toggle the **SendCustom** bit by right clicking the **XIC** and selecting **Toggle Bit** or select the **XIC** and press **Ctrl-T** on Rung 8.

21. Open the **MSG** properties by clicking the ellipsis button on Rung 8.

![Diagram](image)

22. The message configuration tab comes into view. Under the **Service Type** drop down selection you can view all the **Controller Logging Message** types. In this instance, **Controller Log Add Entry** type, we see that **CustomLog** is the source element which is messaged to the controller.
23. Select the communications tab in the CIP message path to the controller. 1 is the default start to any message instruction to point to the backplane of the rack and 0 is the slot of our L85 controller. 

(Note: For CompactLogix controllers are always in slot 0, so the path to a CompactLogix controller is always 1,0).

All of the message instructions were setup by the program when we set the Start bit in the Main Routine. If we want to configure a message to point at a specific controller then select the browse button and point it to our controller.


Let’s create a few log entries to look at when we open our log entry file.

25. In the Online Tool Bar at the top left, select No Forces > I/O Forcing >Enable All I/O Forces then go back and select Disable All I/O Forces in the same manner.
26. Again, in the Online Tool Bar at the top left select **No Forces > SFCForcing > Enable All I/O Forces** then go back and select **Disable All I/O Forces** in the same manner.

![Image](image.png)

27. Minimize Studio 5000 Logix Designer.

28. Right click and open the file **ControllerLogSample.txt** with Notepad or Microsoft Excel contained in **C:\Lab Files\Advanced Logix\Lab 9 Controller Change Log** folder.

Or if you have a SD card reader for you lab computer you can view the log file. It is a .TXT file that is stored in the Logix\Logs directory of the SD card.

**Note:** For lab simplification we may not be able to read an SD card with the lab computer. The file provided is a replica of what we might find after making the changes performed in the lab.

The top left corner specifies the type of file, date of file, type of controller, serial number, and firmware revision of the controller.

<table>
<thead>
<tr>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSV-Controller-Log</td>
</tr>
<tr>
<td>Date = Nov-01-2008 00:22:35</td>
</tr>
<tr>
<td>Controller = 1756-L63/A</td>
</tr>
<tr>
<td>Serial-Number = 16#0016_35C1</td>
</tr>
<tr>
<td>Revision = 17.02</td>
</tr>
</tbody>
</table>
In each controller log entry a sequential record number and time are recorded. An entry description details what occurred. A Windows PC user name is associated with the workstation that the changes were made from. If the plant software is using a FactoryTalk application then a logged in Factory Talk user is also tied to the change. Finally, any extended information like the tolerance values for our custom machine out of tolerance message are shown.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Record Number</td>
<td>Time</td>
<td>Entry Description</td>
<td>User Name</td>
<td>Workstation Name</td>
<td>FactoryTalk Login ID</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Nov-01-2009</td>
<td>10:06:06</td>
<td>Project Imported</td>
<td>PlantDomain:GMall</td>
<td>John Maintenance</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Nov-01-2009</td>
<td>10:06:09</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>Nov-01-2009</td>
<td>10:06:11</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>Nov-01-2009</td>
<td>10:06:14</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>Nov-01-2009</td>
<td>10:06:16</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>Nov-01-2009</td>
<td>10:06:18</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>Nov-01-2009</td>
<td>10:06:21</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>Nov-01-2009</td>
<td>10:06:23</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>Nov-01-2009</td>
<td>10:06:24</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>20</td>
<td>19</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>22</td>
<td>21</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>24</td>
<td>23</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
<tr>
<td>25</td>
<td>24</td>
<td>Nov-01-2009</td>
<td>10:06:27</td>
<td>SPC Limits changed</td>
<td>Local</td>
<td>None</td>
</tr>
</tbody>
</table>

Lab Summary

You should now have a better understanding of how to implement Controller Logging and the features available. Knowing and understanding changes that were made to a controller aid in troubleshooting field issues in a variety of applications.

Congratulations!

You have just finished this Lab

Click to Proceed to Contents