ioCONTROL USER'S GUIDE

Form 1300-060515 —May, 2006

OPTO 22
43044 Business Park Drive • Temecula • CA 92590-3614
Phone: 800-321-OPTO (6786) or 951-695-3000
Fax: 800-832-OPTO (6786) or 951-695-2712
www.opto22.com

Product Support Services
800-TEK-OPTO (835-6786) or 951-695-3080
Fax: 951-695-3017
Email: support@opto22.com
Web: support.opto22.com

ioControl User's Guide
Form 1300-060515 —May, 2006

The information in this manual has been checked carefully and is believed to be accurate; however, Opto 22 assumes no responsibility for possible inaccuracies or omissions. Specifications are subject to change without notice.

Opto 22 warrants all of its products to be free from defects in material or workmanship for 30 months from the manufacturing date code. This warranty is limited to the original cost of the unit only and does not cover installation, labor, or any other contingent costs. Opto 22 I/O modules and solid-state relays with date codes of 1/96 or later are guaranteed for life. This lifetime warranty excludes reed relay, SNAP serial communication modules, SNAP PID modules, and modules that contain mechanical contacts or switches. Opto 22 does not warrant any product, components, or parts not manufactured by Opto 22; for these items, the warranty from the original manufacturer applies. These products include, but are not limited to, OptoTerminal-G70, OptoTerminal-G75, and Sony Ericsson GT-48; see the product data sheet for specific warranty information. Refer to Opto 22 form number 1042 for complete warranty information.

Cyrano, Opto 22 FactoryFloor, Optomux, and Pamux are registered trademarks of Opto 22. Generation 4, ioControl, ioDisplay, ioManager, ioProject, ioUtilities, mistic, Nvio, Nvio.net Web Portal, OptoConnect, OptoControl, OptoDisplay, OptoENETSniff, OptoOPCServer, OptoScript, OptoServer, OptoTerminal, OptoUtilities, SNAP Ethernet I/O, SNAP I/O, SNAP OEM I/O, SNAP Simple I/O, SNAP Ultimate I/O, and SNAP Wireless LAN I/O are trademarks of Opto 22.

ActiveX, JScript, Microsoft, MS-DOS, VBScript, Visual Basic, Visual C++, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries. Linux is a registered trademark of Linus Torvalds. Unicenter is a registered trademark of Computer Associates International, Inc. ARCNET is a registered trademark of Datapoint Corporation. Modbus is a registered trademark of Schneider Electric. Wiegand is a registered trademark of Sensor Engineering Corporation. Nokia, Nokia M2M Platform, Nokia M2M Gateway Software, and Nokia 31 GSM Connectivity Terminal are trademarks or registered trademarks of Nokia Corporation. Sony is a trademark of Sony Corporation. Ericsson is a trademark of Telefonaktiebolaget LM Ericsson.

All other brand or product names are trademarks or registered trademarks of their respective companies or organizations.

Table of Contents

Chapter 8: Working with Flowcharts....8-1

CHAPTER 1

Welcome to ioControl

Introduction

Welcome to ioControl™, Opto 22's visual control language for Opto 22 Ethernet-based control systems and the input/output (I/O) units that work with them.

ioControl makes it easy to write control applications with little or no programming experience. If you know how to design a control application and can draw some flowcharts to describe it, you already know the basics of ioControl. At the same time, ioControl provides a complete and powerful set of commands, as well as the OptoScript™ programming language, to meet your most demanding industrial control needs.

ioControl comes in two forms: ioControl Basic and ioControl Professional™.

- **ioControl Basic** is included in the purchase of an Opto 22 Ethernet-based controller and is also available as a free download from our Web site, www.opto22.com. ioControl Basic includes both flowchart and OptoScript programming, subroutines, a graphical debugger, and about 400 commands.
- **ioControl Professional** is available for purchase either separately or as part of the complete ioProject Professional™ software suite. The Professional version of ioControl adds Ethernet link redundancy to controllers and I/O units, supports additional features in Ethernet-based I/O units (such as ramping and pulsing), and offers additional data types in subroutines. ioControl Professional also provides a migration path for Opto 22 FactoryFloor™ customers by adding support for serial-based *mistic*® I/O units and by including a conversion utility to help customers move older OptoControl™ strategies to ioControl. ioControl Professional includes about 500 commands.

About this Guide

This user's guide not only teaches you how to use ioControl, but also provides programming instruction and tips. The separate *ioControl Command Reference* describes in detail all ioControl programming commands, or instructions.

This guide assumes that you are already familiar with Microsoft® Windows® on your personal computer, including how to use a mouse, standard commands, and menu items to open, save, and close files. If you are not familiar with Windows or your PC, refer to the documentation from Microsoft and your computer manufacturer.

This guide covers both ioControl Basic and ioControl Professional. Features that are available only in ioControl Professional are marked with Pro .

Here's what is in this user's guide:

Chapter 1, "Welcome to ioControl"—Information about the guide and how to reach Opto 22 Product Support.

Chapter 2, "ioControl Tutorial"—A tutorial designed to help you use ioControl as quickly as possible. The chapter leads you through a sample strategy that you can manipulate, download, and run in Debug mode.

Chapter 3, "What Is ioControl?"—An introduction to ioControl, key terminology, and the main windows and toolbars.

Chapter 4, "Designing Your Strategy"—Programming in ioControl: how to get from your real-world control problem to a working strategy.

Chapter 5, "Working with Control Engines"—How to configure and communicate with control engines.

Chapter 6, "Working with I/O"—How to configure and communicate with input/output (I/O) units, I/O points, and PID loops.

Chapter 7, "Working with Strategies"—Detailed steps for creating, compiling, and running strategies.

Chapter 8, "Working with Flowcharts"—Detailed steps for creating and working with the flowcharts that make up your strategy.

Chapter 9, "Using Variables and Commands"—Steps for configuring the seven types of variables you can use in programming: communication handle, numeric, string, pointer, numeric table, string table, and pointer table variables. Also shows how to use the commands that control the I/O and variables you've configured.

Chapter 10, "Programming with Commands"—Important tips on using ioControl commands to accomplish what you want in your strategy.

Chapter 11, "Using OptoScript"—Details on the optional scripting language available in ioControl for complex loops, string handling, and mathematical expressions.

Chapter 12, "Using Subroutines"—How to use subroutines to streamline your strategy development.

Appendix A, "ioControl Troubleshooting"—Tips for resolving communication problems and other difficulties you may encounter.

Appendix B, "ioControl Errors and Messages"—Types of errors, where you'll see them, and the causes of common errors.

Appendix C, "ioControl Files"—A list of all ioControl files located in the ioControl directory and in any strategy directory.

Appendix D, "Sample Strategy"—An illustration and description of the sample "Cookies" strategy used in Chapter 1.

Appendix E, "OptoScript Command Equivalents"—A table of all standard ioControl commands, showing their equivalents in OptoScript code.

Appendix F, "OptoScript Language Reference"—Details about OptoScript code, including comparisons to other languages, lexical reference, and notes to experienced programmers.

Index—Alphabetical list of key words and the pages they are located on.

Document Conventions

The following conventions are used in this document:

- The Pro icon next to text indicates that a feature is available only in ioControl Professional, not in ioControl Basic.
- Italic typeface indicates emphasis and is used for book titles. (Example: "See the *ioDisplay User's Guide* for details.")
- Names of menus, commands, dialog boxes, fields, and buttons are capitalized as they appear in the product. (Example: "From the File menu, select Print.")
- File names appear either in all capital letters or in mixed case, depending on the file name itself. (Example: "Open the file TEST1.txt.")
- Key names appear in small capital letters. (Example: "Press SHIFT.")
- Key press combinations are indicated by hyphens between two or more key names. For example, SHIFT+F1 is the result of holding down the shift key, then pressing and releasing the F1 key. Similarly, CTRL+ALT+DELETE is the result of pressing and holding the CTRL and ALT keys, then pressing and releasing the DELETE key.
- "Click" means press and release the left mouse button on the referenced item. "Right-click" means press and release the right mouse button on the item.
- Menu commands are referred to with the Menu→Command convention. For example, "File→Open Project" means to select the Open Project command from the File menu.
- Numbered lists indicate procedures to be followed sequentially. Bulleted lists (such as this one) provide general information.

Other Resources

Documents and Online Help

To help you understand and use ioControl systems, the following resources are provided:

- **Online Help** is available in ioControl and in most of the utility applications. To open online Help, choose Help→Contents and Index in any screen.
- ***ioControl User's Guide*** shows how to install and use ioControl.
- ***ioControl Command Reference*** contains detailed information about each command (instruction) available in ioControl.
- A **quick reference card**, located in the front pocket of the *ioControl Command Reference*, lists all ioControl commands plus their OptoScript™ code equivalents and arguments.
- ***ioManager User's Guide*** and other guides provided with specific hardware help you install, configure, and use controllers and I/O units.

Online versions (Adobe® Acrobat® format) of ioControl documents are provided on the CD that came with your controller or purchase of Professional software and are also available from the Help menu in ioControl. To view a document, select Help→Manuals, and then choose a document from the submenu.

When you purchase ioControl Professional or ioProject Professional, you also receive a complete set of printed documents.

Resources are also available on the Opto 22 Web site at www.opto22.com. You can conveniently access the Web site using the Help menu in ioControl. Select Help→Opto 22 on the Web, and then select an online resource from the submenu.

Product Support

If you have any questions about ioControl, you can call, fax, or email Opto 22 Product Support.

Phone:	800-TEK-OPTO (835-6786) 951-695-3080 (Hours are Monday through Friday, 7 a.m. to 5 p.m. Pacific Time)
Fax:	951-695-3017
Email:	support@opto22.com
Opto 22 Web site:	support.opto22.com

NOTE: Email messages and phone calls to Opto 22 Product Support are grouped together and answered in the order received.

When calling for technical support, be prepared to provide the following information about your system to the Product Support engineer:

- Software and version being used
- Firmware versions
- PC configuration (type of processor, speed, memory, and operating system)
- A complete description of your hardware and operating systems, including:
  - type of power supply
  - types of I/O units installed
  - third-party devices installed (for example, barcode readers)
- Specific error messages seen.

Installing ioControl

ioControl installation is easy and quick. Insert the CD containing ioControl in your CD-ROM drive, and the installation wizard should appear. If the wizard does not appear, start Windows Explorer and navigate to your CD-ROM drive. Double-click Setup.exe to begin installation.

NOTE: If you run ioProject applications in Microsoft Windows XP, make sure to use the Windows Classic theme. Otherwise, a Microsoft bug with how themes are handled may cause the system to crash.

If you have trouble installing ioControl, contact Opto 22 Product Support at 800-835-6786 or 951-695-3080.

System Requirements

Installation Requirements

Here's what you need to install and run ioControl:

- A computer with at least the minimum processor required for your version of Microsoft Windows (1 GHz Pentium®-class or better recommended) and Ethernet capability
- VGA or higher resolution monitor (Super VGA recommended). Minimum size: 800x600 with small fonts.
- Mouse or other pointing device
- Installed Windows printer (optional)
- Microsoft Windows XP or Windows 2000® (with SP4) workstation operating system
- At least 128 MB RAM (256 MB recommended)

  If you are using ioDisplay, please note its requirements in the *ioDisplay User's Guide*.
- Available hard disk space : at least 29 MB for ioControl Basic or at least 30 MB for ioControl Professional
- Compatible control engine and I/O unit(s), as shown in the following section.

Compatible Control Engines and I/O Units

The following control engine and I/O unit combinations are compatible with ioControl Basic and ioControl Professional as shown:

Using this control engine	ioControl Basic supports these I/O units	ioControl Professional supports these I/O units
SNAP PAC S-series controller	SNAP-PAC-R1 SNAP-PAC-R2 SNAP Ultimate I/O SNAP Ethernet I/O SNAP Simple I/O E1 and E2*	SNAP-PAC-R1 SNAP-PAC-R2 SNAP Ultimate I/O SNAP Ethernet I/O SNAP Simple I/O E1 and E2* B3000 serial SNAP-BRS G4D16R, G4D32RS, G4A8R B100 and B200
SNAP PAC R-series controller	SNAP-PAC-R1 SNAP-PAC-R2 SNAP Ultimate I/O SNAP Ethernet I/O SNAP Simple I/O E1 and E2*	SNAP-PAC-R1 SNAP-PAC-R2 SNAP Ultimate I/O SNAP Ethernet I/O SNAP Simple I/O E1 and E2*
SNAP-LCE controller	SNAP-PAC-R1 SNAP-PAC-R2 SNAP Ultimate I/O SNAP Ethernet I/O SNAP Simple I/O E1 and E2*	**
SNAP Ultimate controller/brain	SNAP-PAC-R1 SNAP-PAC-R2 SNAP Ultimate I/O SNAP Ethernet I/O SNAP Simple I/O E1 and E2*	**

* E1 and E2 I/O units are supported for the features available under the OptoMMP protocol only, not for all Optomux features. See the *E1 and E2 User's Guide* (form #1563) for details. For configuration, see *I/O Configuration for E1 and E2 Brain Boards* (form #1576).

** SNAP-LCE controllers and SNAP Ultimate I/O can use ioControl Basic, but they are not equipped to use the extra features in ioControl Professional. For example, they do not have dual Ethernet interfaces for network segmenting nor RS-485 ports to support serial mistic I/O units.

Important Note on Disk Drives

Opto 22 applications, including ioControl, perform best when using files from a local hard disk. Network drives may be used, but performance may suffer and depends upon the speed and reliability of the network. While it may be possible to use other drive types, such as floppy disks, key chain USB drives, and memory cards, their use is not recommended. They are better suited for transferring files rather than directly accessing them.

CHAPTER 2

ioControl Tutorial

Introduction

In this chapter, we'll start with a sample strategy: a control application for a simple cookie factory. You'll learn how to work with strategies, open and manipulate flowcharts, work with variables and I/O points, configure a control engine, compile and download a strategy, run it in Debug mode, make an online change, and more. The tutorial can be used with either ioControl Basic or ioControl Professional. (ioControl Basic is shown in the graphics.)

The best way to use the tutorial is to sit down at your computer and follow it through. All you need is ioControl and access to a control engine. Even if you can't access a control engine at the moment, you can still do everything up to the point of downloading your strategy.

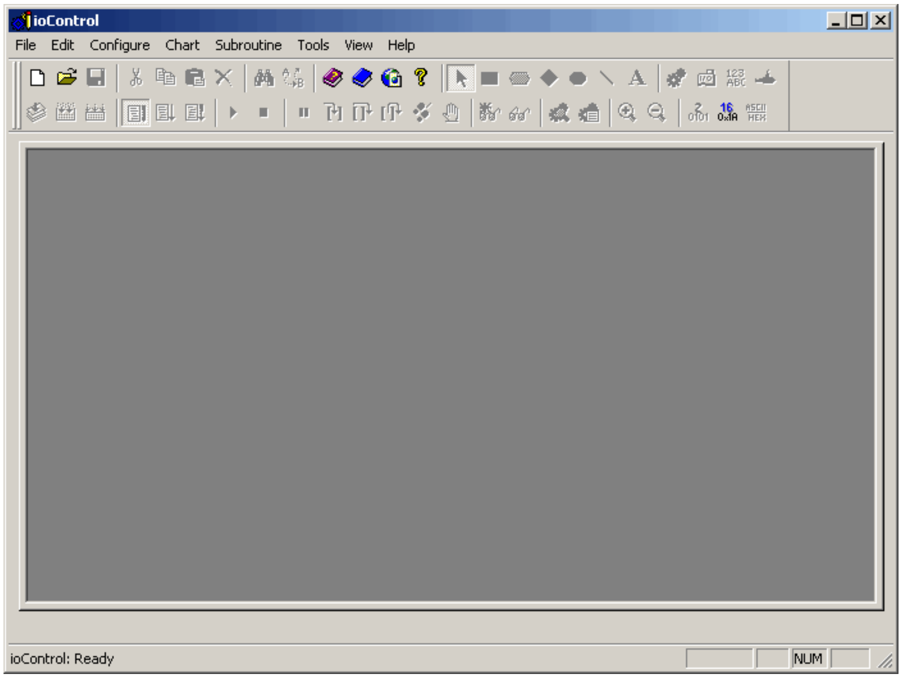
In this Chapter

Opening the Strategy

A strategy is a complete control program developed in ioControl. Our sample strategy controls a cookie factory. Appendix C describes the sample strategy in detail, but for now, let's just open and explore it.

1. Start ioControl by clicking the Start button and selecting Programs→Opto 22→ioProject Software→ioControl.

The ioControl main window opens.

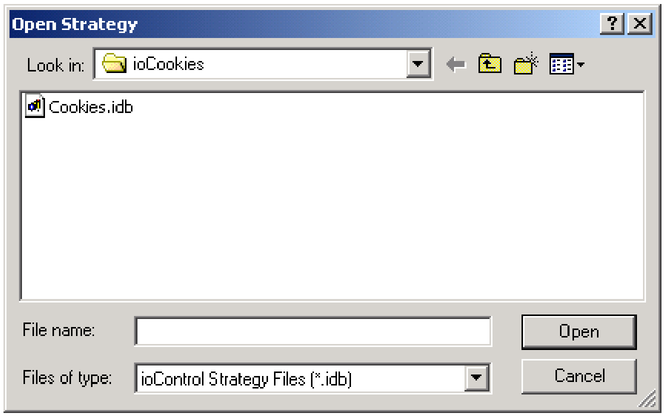


2. Click the Open Strategy button on the toolbar, or choose File→Open Strategy.

3. In the Open Strategy dialog box, navigate to ioControl\Examples.

4. In the Examples directory, double-click the ioCookies subdirectory to open it.

   The strategy file Cookies.idb appears.



Look in: ioCookies

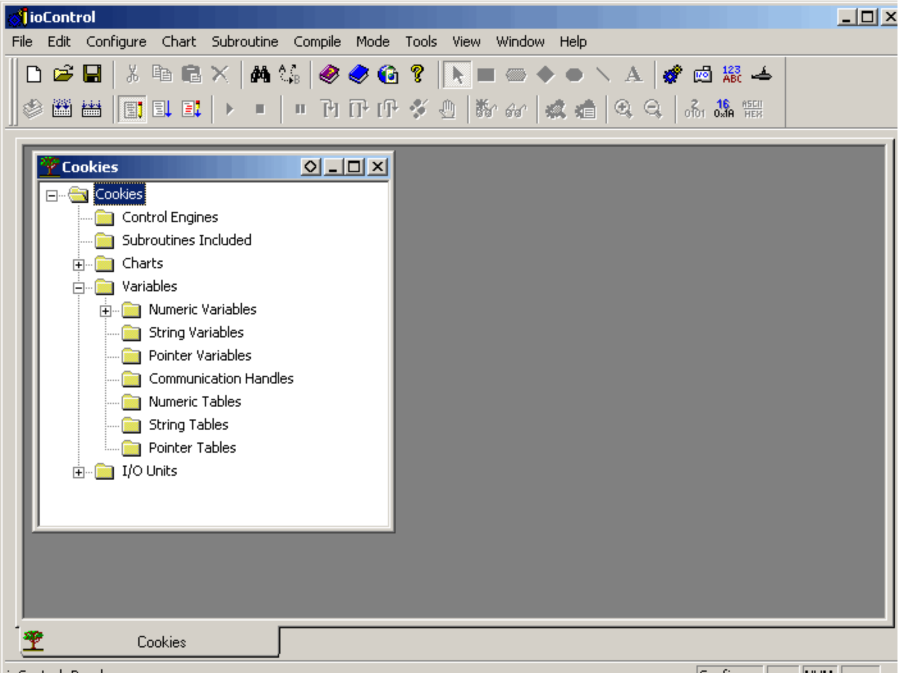
Cookies.idb

File name:

Files of type: ioControl Strategy Files (*.idb)

5. Double-click the Cookies.idb file to open it.

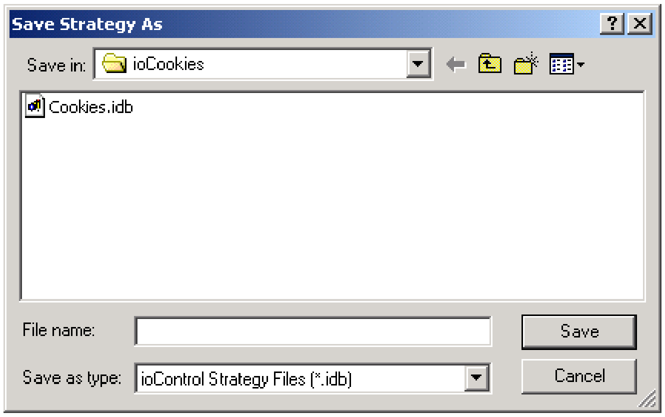
The Cookies strategy opens and the ioControl window now shows the Cookies strategy (yours may look somewhat different).



Saving the Strategy

Now let's save the strategy to a new name, so we can change it while leaving the original intact.

1. Select File→Save Strategy As.

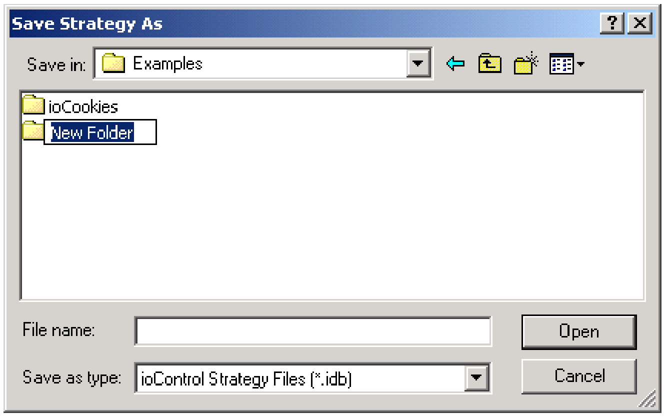


Since each ioControl strategy must be located in its own directory, we cannot save the strategy to a new name in its current location.

2. Click the Up One Level button to move up to the Examples directory.

3. Click the Create New Folder button .

   The new folder appears in the list.



Save Strategy As

Save in: Examples

ioCookies

New Folder

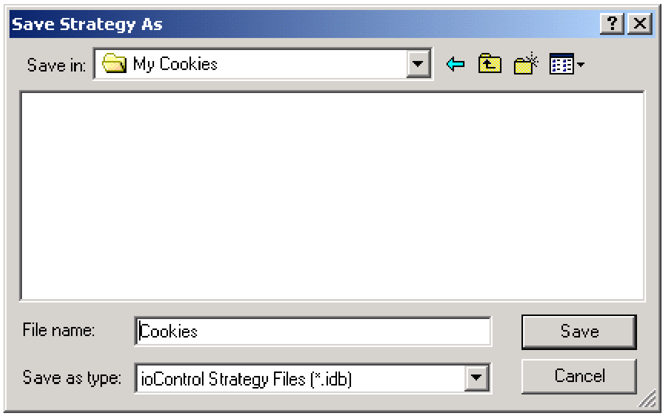
File name:

Save as type: ioControl Strategy Files (*.idb)

4. Type `My Cookies` to replace New Folder. Double-click the folder to open it.

5. Click in the File name field and type `Cookies`.

   The dialog box now looks like this:



Save Strategy As

Save in: My Cookies

File name: Cookies

Save as type: ioControl Strategy Files (*.idb)

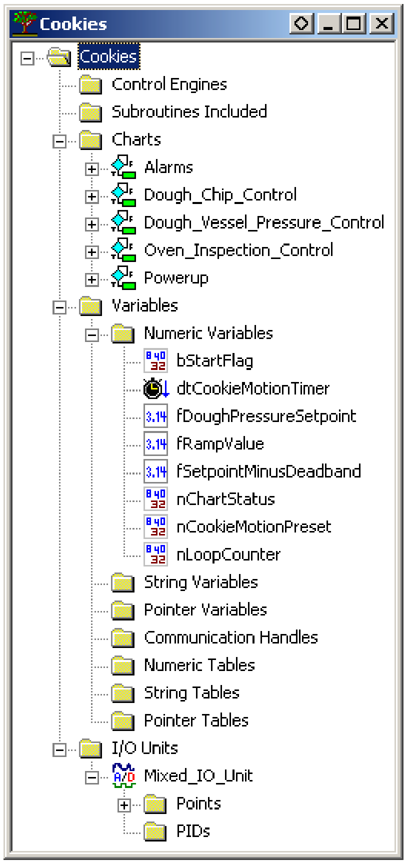
6. Click Save.

   The strategy is saved as Cookies in the My Cookies directory.

Examining the Strategy

Briefly, our cookie factory includes a tank of pre-mixed cookie dough, a tank of chocolate chips, an oven, a visual inspection station, a conveyor belt, and some compressed air to blow rejected cookies off the belt. The process starts when a ball of dough drops on the belt. It moves along under the chip tank to receive some chips, and then it moves into the oven to be baked. The next stop is an inspection, where rejected cookies are blown off the belt and good cookies move along to shipping. Should anything go wrong, we also have some alarms built in to stop the process when necessary.

The best way to see all the components of the strategy is to look at the Strategy Tree.



The Strategy Tree

As with any window in ioControl, you can move the Strategy Tree window by clicking and dragging the title bar, you can minimize, maximize, or dock it by clicking buttons at the right of the title bar, or you can reshape it by dragging any edge in any direction.

However, the Strategy Tree window is unique in that it must remain open, since closing it is equivalent to closing the strategy.

The Strategy Tree works like Windows Explorer: you can expand and collapse folders to show or hide what is in them. A quick look at the tree reveals that our strategy includes five flowcharts (in the Charts folder), eight Numeric Variables, and one Mixed I/O Unit (with both analog and digital points).

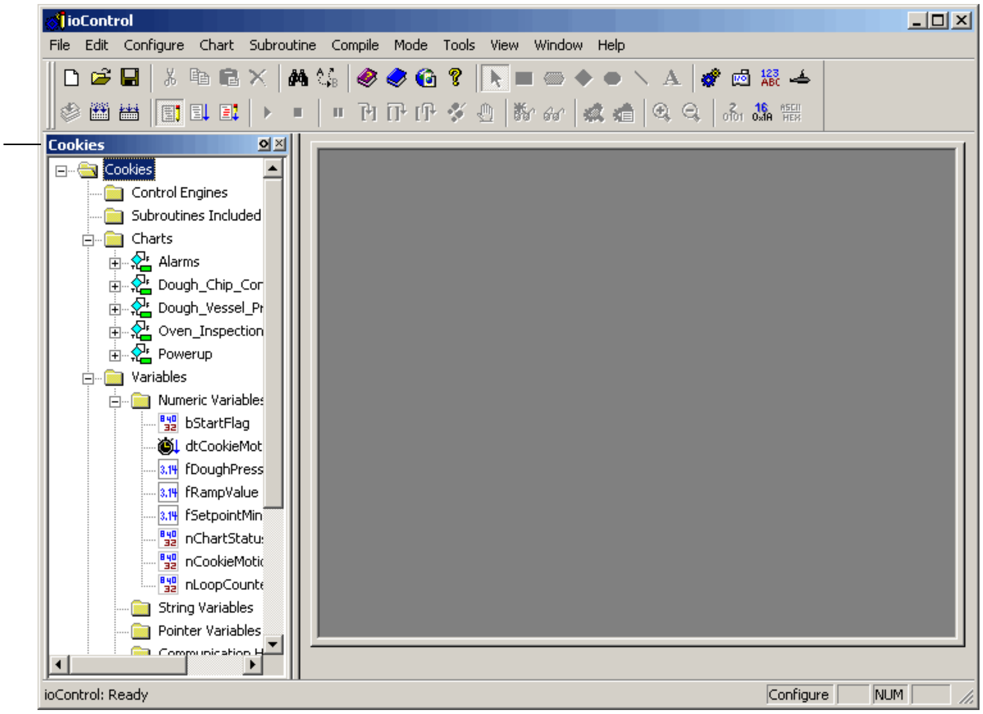
The Strategy Tree not only shows you all components of the strategy but also provides shortcuts to many common ioControl activities, for example, opening flowcharts.

Docking the Strategy Tree

Since the Strategy Tree is so useful, you'll probably want to keep it visible while you create and debug your strategy. To keep the Strategy Tree window always visible, you can dock it in a separate frame.

1. Click the docking button in the upper-right corner of the Strategy Tree window.

   The Strategy Tree moves into its own frame at the left side of the main window.



Docked Strategy Tree

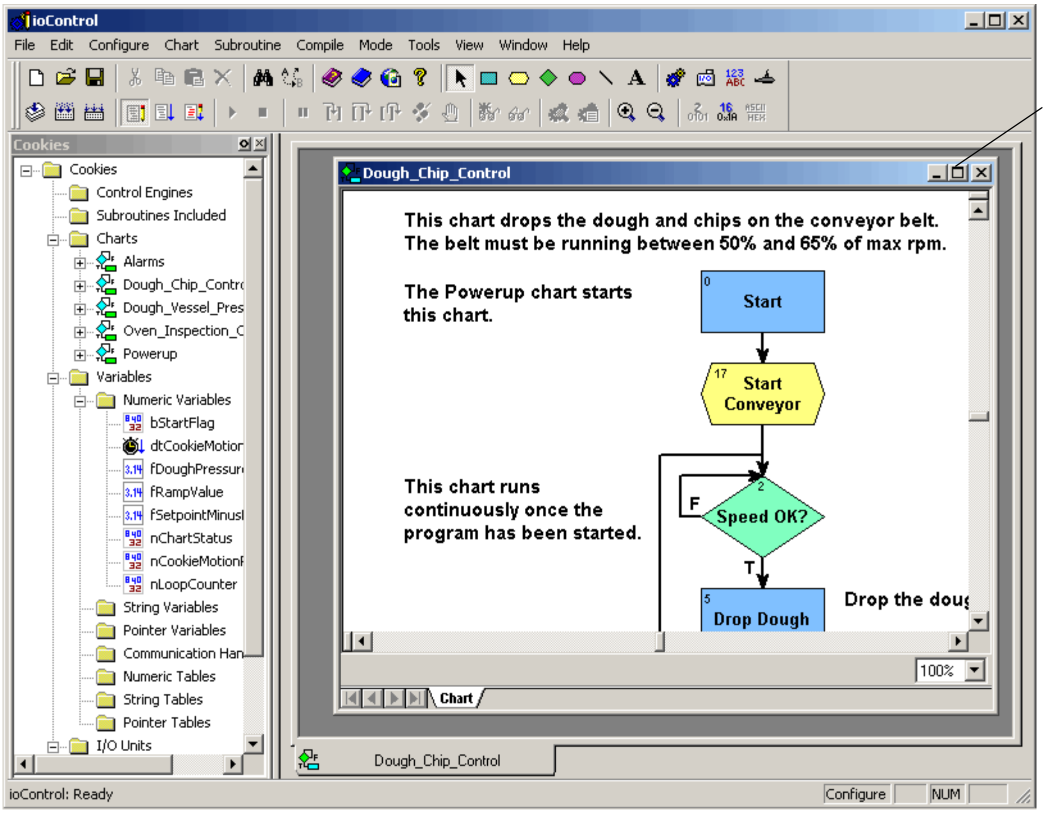
2. To change the width of the Strategy Tree's frame, move your mouse over the right side of the frame. When the cursor changes, click and drag the side to make the frame wider or narrower.

Opening a Chart

Every control process can be planned and mapped out using one or several ioControl flowcharts, or charts. Because flowcharts are easy to follow, ioControl strategies are easy to understand. For an example, let's take a look at the Dough_Chip_Control chart.

1. Double-click the Dough_Chip_Control chart name on the Strategy Tree. (You can also open a chart by selecting Chart➞Open and navigating to the chart name.)

   The chart appears somewhere in the large frame of the main window.



This chart drops the dough and chips on the conveyor belt. The belt must be running between 50% and 65% of max rpm.

The Powerup chart starts this chart.

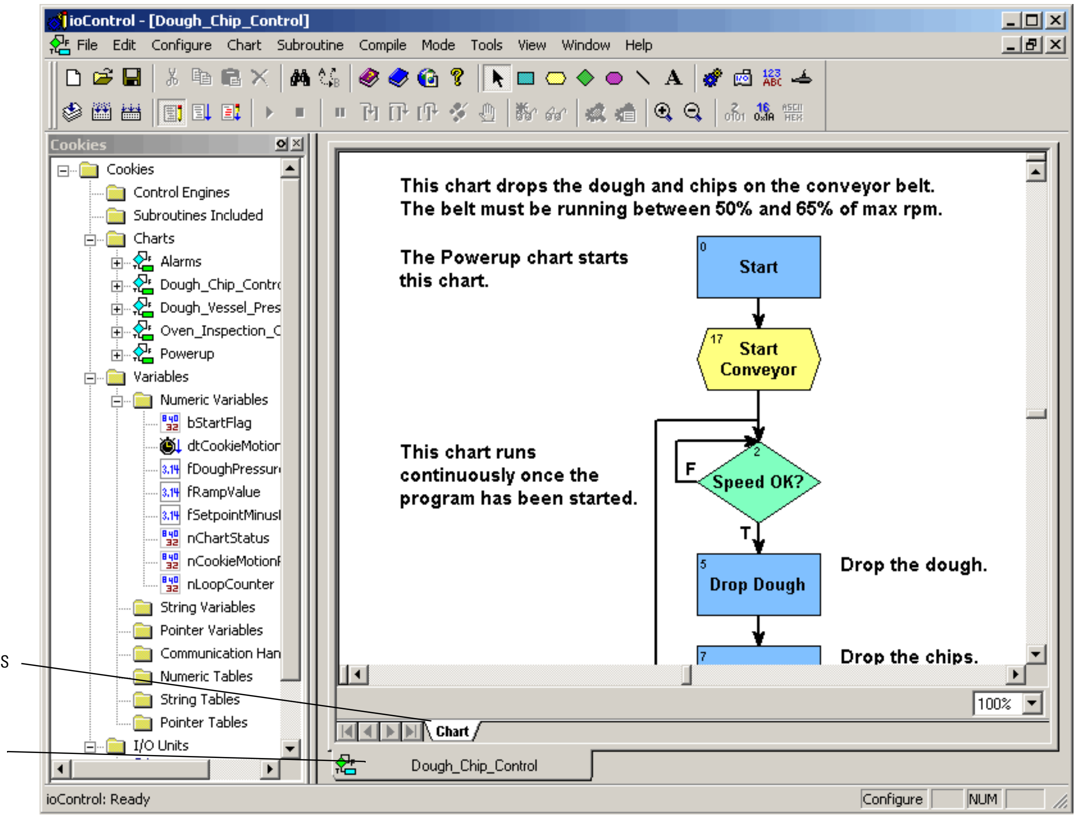
This chart runs continuously once the program has been started.

Drop the doug

Maximize button

2. Click and drag the title bar of the chart window if necessary to see the maximize button at the upper right. Click the maximize button.

The chart now covers the whole frame. Notice the tabs at the bottom of the main window; the white tab tells you you're viewing a chart, and the gray tab shows the chart's name.



This chart drops the dough and chips on the conveyor belt. The belt must be running between 50% and 65% of max rpm.

The Powerup chart starts this chart.

This chart runs continuously once the program has been started.

Drop the dough.

Drop the chips.

White tab shows this is a chart.

Gray tab shows chart name.

Let's take a closer look at what this chart does. Even without its descriptive comments, it's easy to see that this program segment begins by starting a conveyor belt. If the belt is not running at the correct speed, the process goes into a loop until the speed is correct. When it is correct, dough is dropped on the belt, and then chips are dropped on the dough. The process then loops back to re-evaluate the conveyor speed, waiting if it's incorrect, and dropping more dough and chips if it's correct.

The rectangular-shaped blocks are called action blocks. They do things. The diamond-shaped blocks are condition blocks. They decide things. Charts may also contain other blocks, including oval-shaped continue blocks, which route the program logic back to another block in the same chart, and hexagon-shaped script blocks, which contain instructions and logic written in ioControl's built-in OptoScript programming language.

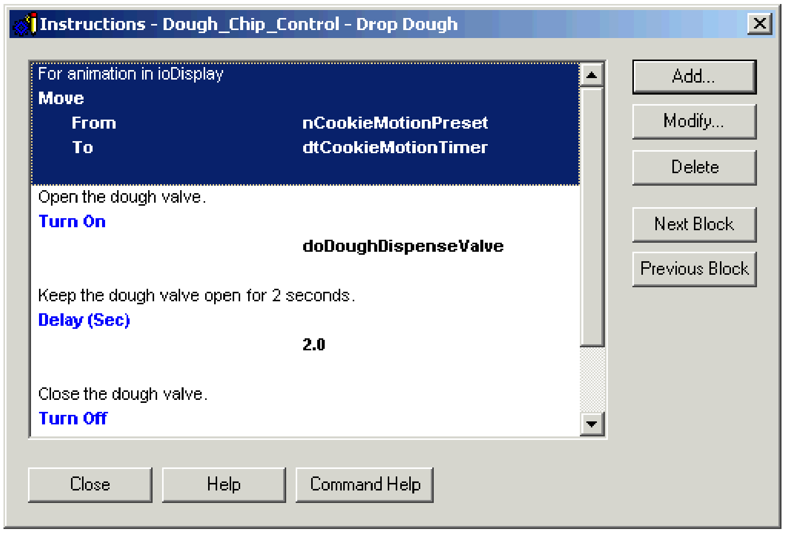
Connections link the blocks together and show how the program logic flows. Action blocks exit through just one connection, since they always go in one direction. Condition blocks exit through two connections, one for a true evaluation and the other for a false evaluation.

Opening a Block

Let's see what's in a block.

1. Double-click the Drop Dough block.

   The Instructions dialog box appears.



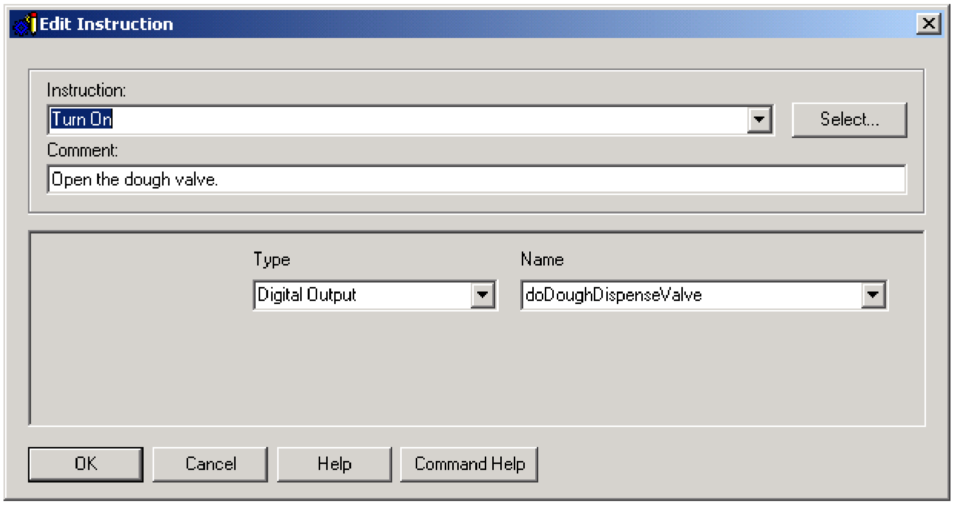
Instructions - Dough_Chip_Control - Drop Dough

```
For animation in ioDisplay
Move
     From                    nCookieMotionPreset
     To                      dtCookieMotionTimer
Open the dough valve.
Turn On
                             doDoughDispenseValve

Keep the dough valve open for 2 seconds.
Delay (Sec)
                             2.0

Close the dough valve.
Turn Off
```

This block contains four instructions: Move, Turn On, Delay (Sec), and Turn Off. Each one has a description above it.

2. Double-click the Turn On instruction to see more about it. (You could also click it once and click Modify).

   The Edit Instruction dialog box for the Turn On command appears.



Edit Instruction

Instruction: Turn On

Comment: Open the dough valve.

Type	Name
Digital Output	doDoughDispenseValve

Here we see the details of the command, which simply turns on the digital output Dough_Dispense_Valve. In other words, it opens the valve.

3. Close the Edit Instruction dialog box by clicking OK or Cancel.

4. Back in the Drop Dough block's Instructions dialog box, move your cursor to the bottom right edge of the dialog box.

   When your cursor changes to a bidirectional arrow, you can resize this dialog box by clicking and dragging any edge.

5. Close the Instructions - Dough_Chip_Control - Drop Dough dialog box by clicking Close or by pressing ESC.

   Before we leave the Dough_Chip_Control chart, let's make a cosmetic change. We noted earlier that we didn't have any continue blocks in this chart. Let's add one to replace the long connection that loops from the Drop Chips block up to the Speed OK? block.

6. Select the connection line by clicking it at a bend or junction point, and delete it by pressing DELETE. Click the down arrow at the bottom right of the chart window once to scroll down a little.

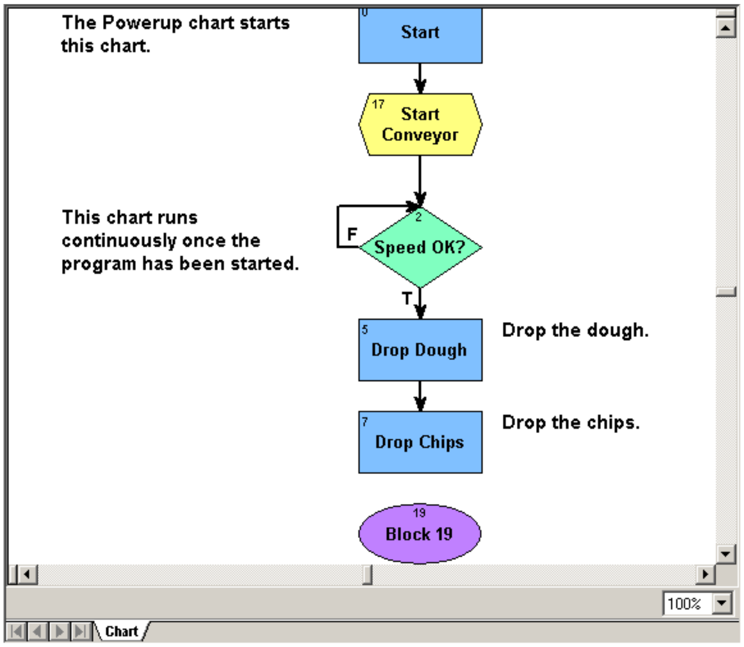
7. Now click the continue block tool button in the toolbar.

   When you bring the mouse back into the chart area, an oval outline appears, representing the new continue block.

8. Position the oval about half an inch below the Drop Chips block, and click your mouse button once.

   A continue block appears with the default name Block 19. (The number on yours may be different.) If you move the mouse again, a new oval outline follows. Deactivate the continue block tool by clicking the right mouse button or by pressing ESC.

   Your screen should now look like this:



The Powerup chart starts this chart.

This chart runs continuously once the program has been started.

Drop the dough.

Drop the chips.

9. Connect the Drop Chips block to this new block by clicking the connection tool ╲ in the toolbar. Click once in the block the connection is coming from, Drop Chips.

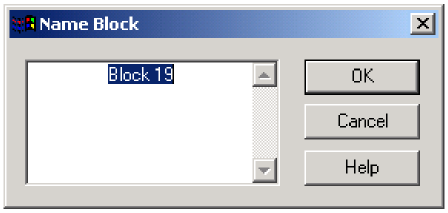
If you move your cursor around the screen, you see a connection following your movements. If you move the cursor to the top of Block 19, the connection becomes a short line from the bottom center of Drop Chips to the top center of Block 19.

10. Click inside the upper part of Block 19 to complete the connection. Click the right mouse button or press ESC to release the tool, returning your cursor to an arrow.

Let's name Block 19 something more descriptive.

11. Click Block 19 once to select it, then right-click it and select Name from its pop-up menu.

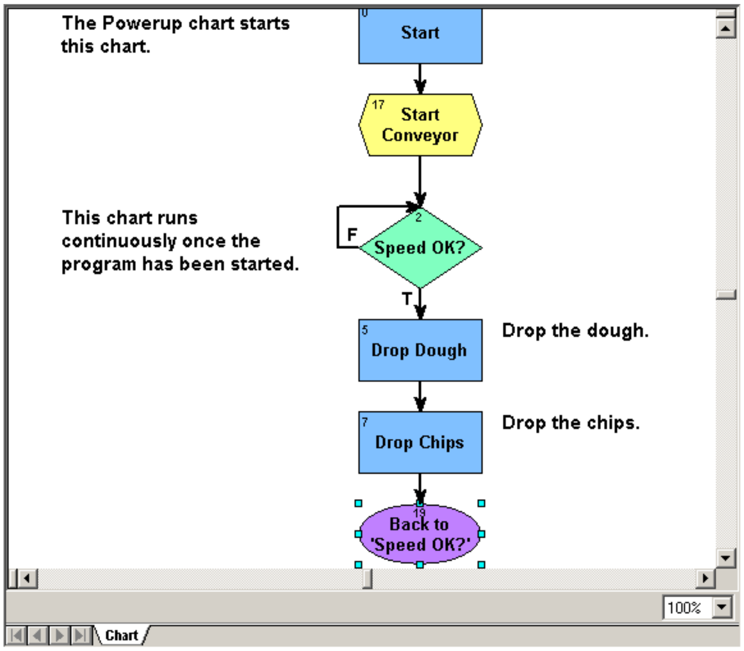
The Name Block dialog box appears.



Block 19

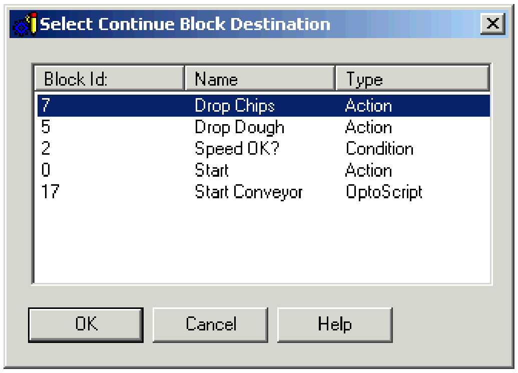
12. Type `Back to 'Speed OK?'` right over the selected text, then click OK.

The chart now looks like this:



Now let's give the continue block its instructions.

13. Double-click the continue block to see a list of all the blocks in the chart.



Block Id:	Name	Type
7	Drop Chips	Action
5	Drop Dough	Action
2	Speed OK?	Condition
0	Start	Action
17	Start Conveyor	OptoScript

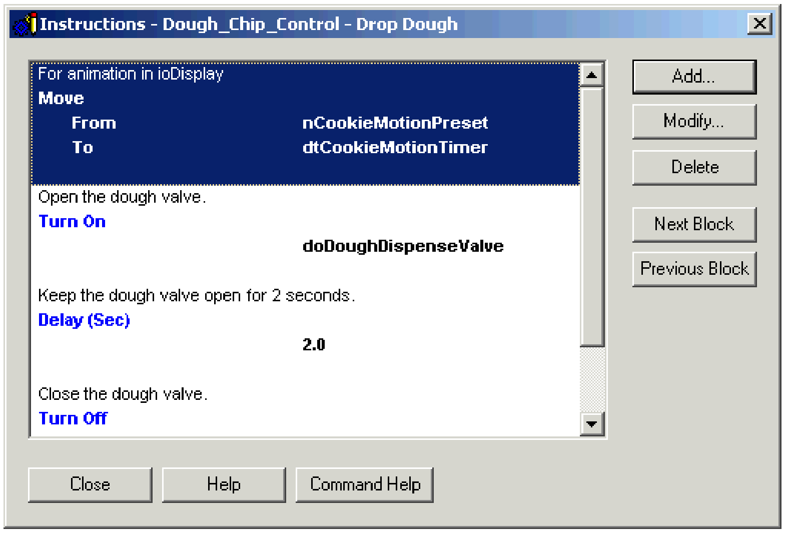
14. Select the Speed OK? block by clicking it once, and then click OK.

When the program reaches the continue block, it returns to the Speed OK? block. We haven't changed the way the program works; we've just done the same thing in a different way. Continue blocks are useful in a complex chart to avoid crossing connection lines.

Adding a Command

Now we're going to add a command, or instruction, to a block in the Dough_Chip_Control chart. We'll add the command so we can keep track of the number of cookies we produce.

1. Double-click the Drop Dough block.



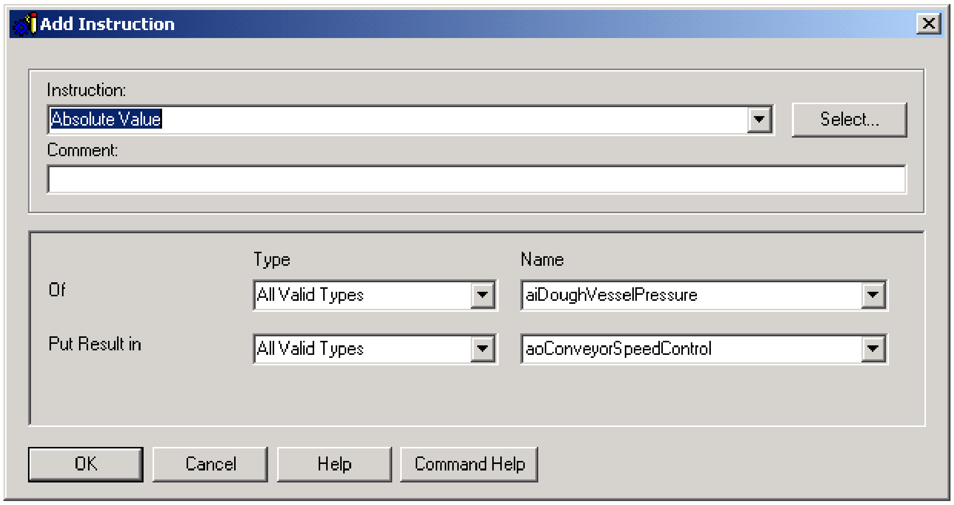
```
For animation in ioDisplay
Move
    From                    nCookieMotionPreset
    To                      dtCookieMotionTimer
Open the dough valve.
Turn On
                            doDoughDispenseValve
Keep the dough valve open for 2 seconds.
Delay (Sec)
                            2.0
Close the dough valve.
Turn Off
```

We'll add the new instruction (command) between the Turn On and Delay (Sec) commands.

2. Click anywhere on Delay (Sec) to highlight this command.

This highlight indicates the position where the next command is added.

3. Click Add to open the Add Instruction dialog box.



Instruction: Absolute Value

Comment:

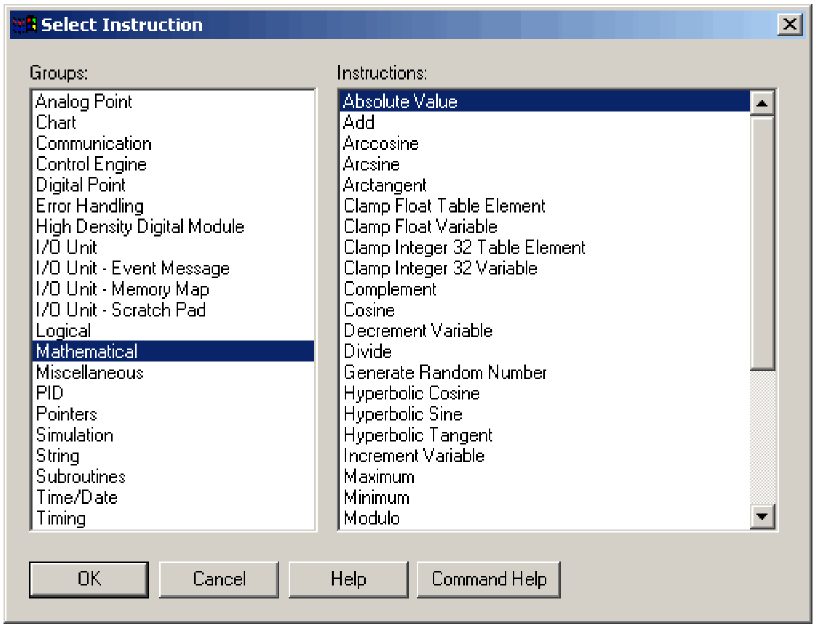
	Type	Name
Of	All Valid Types	aiDoughVesselPressure
Put Result in	All Valid Types	aoConveyorSpeedControl

If you knew the exact name of the command to enter, you could type it over Absolute Value (which is first in the alphabetical list of commands). As you typed, the first command that matched the pattern you entered would be filled in automatically. For example, if you really wanted to enter Absolute Value, you would start typing `abs`.

Another way to add a command is to click the down arrow just to the left of the Select button. You could scroll through the resulting list of commands to find the right one.

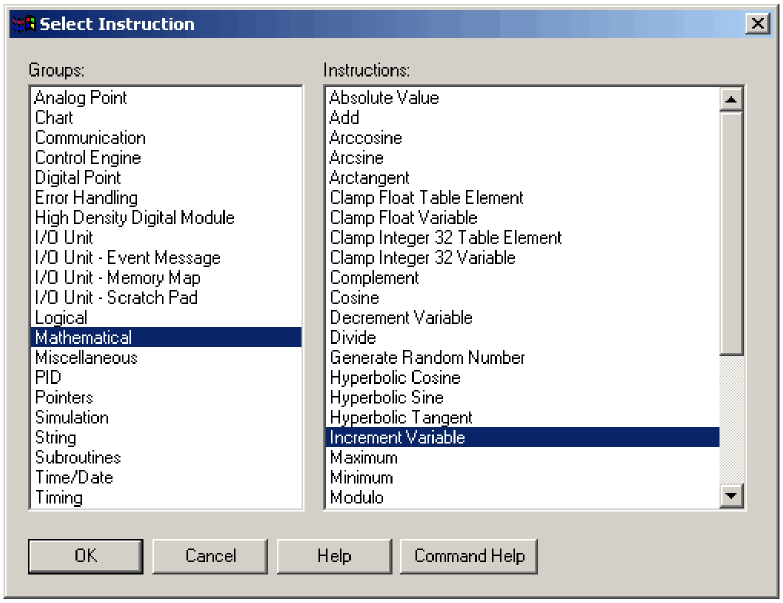
The third way to enter a command is the one we'll use here.

4. Click Select.



All ioControl command groups are listed on the left, and commands in the highlighted group are listed on the right. The command we want has something to do with increasing a counter. Counting sounds like math, so let's try the Mathematical group.

5. Click Mathematical on the left to highlight it, then scroll down the list on the right until you find the command Increment Variable. Click it once.



Notice that if you need information about any command, you can click the Command Help button.

6. Click OK and this command is entered in the Add Instruction dialog box.

The cursor is automatically moved to the next field, which is Comment. Comments are optional; they can help someone else understand the purpose of the instruction.

7. In the Comment field, type `Increment a counter of cookies produced`.

8. Next, click the arrow in the Type field, the one that currently reads All Valid Types.

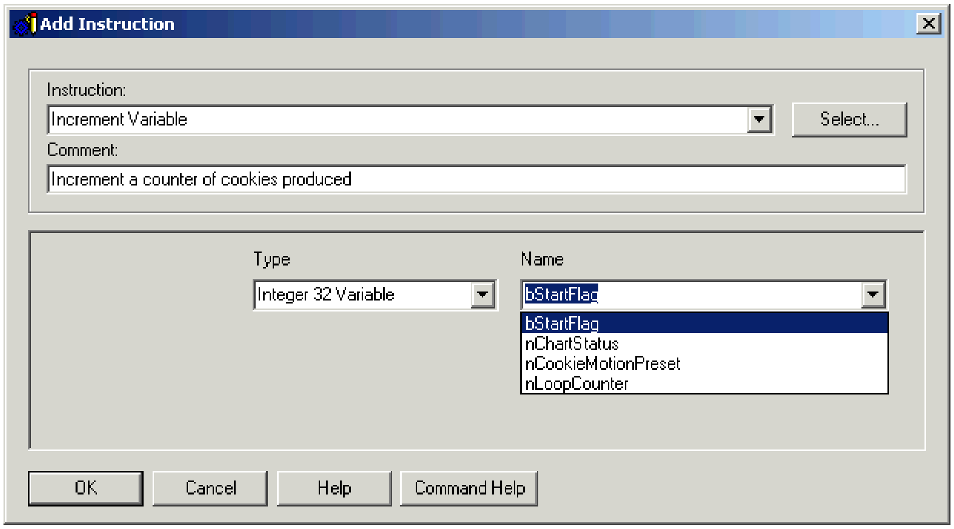
This list shows what those valid types are: a float variable, an integer 32 variable, and an integer 64 variable.

9. Counters are integers, so select Integer 32 Variable.

Now we're going to add the integer 32 variable to increment, which will be called nCookie_Counter.

10. Click the arrow in the Name field, which currently reads bStartFlag.

The drop-down list shows all variables currently configured as integer 32 variables:



Instruction: Increment Variable

Comment: Increment a counter of cookies produced

Type: Integer 32 Variable

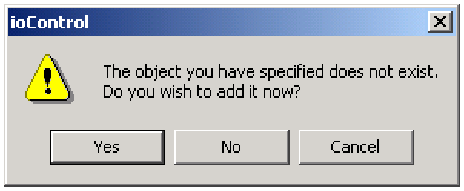
Name: bStartFlag

bStartFlag
nChartStatus
nCookieMotionPreset
nLoopCounter

nCookie_Counter is not in the list, because we never needed it before, so we never created it. We'll create it now using what we call on-the-fly configuration.

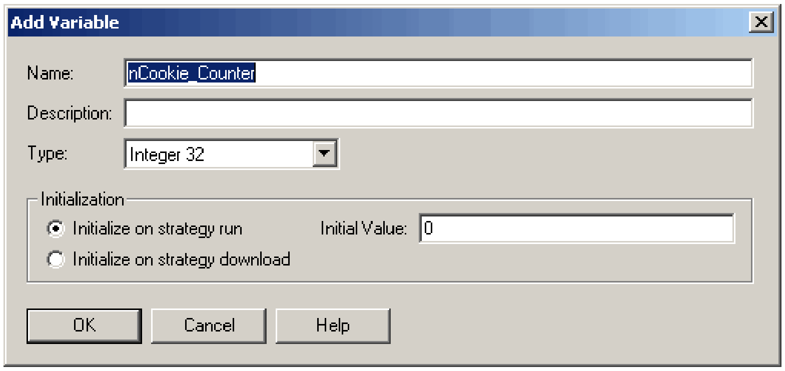
11. Highlight bStartFlag and type the new variable name, nCookie_Counter, right over it.

As soon as you try to do something else, such as click OK to close the dialog box, the following message appears.



The object you have specified does not exist.
Do you wish to add it now?

12. Click Yes.



Name: nCookie_Counter

Description:

Type: Integer 32

Initialization

● Initialize on strategy run

○ Initialize on strategy download

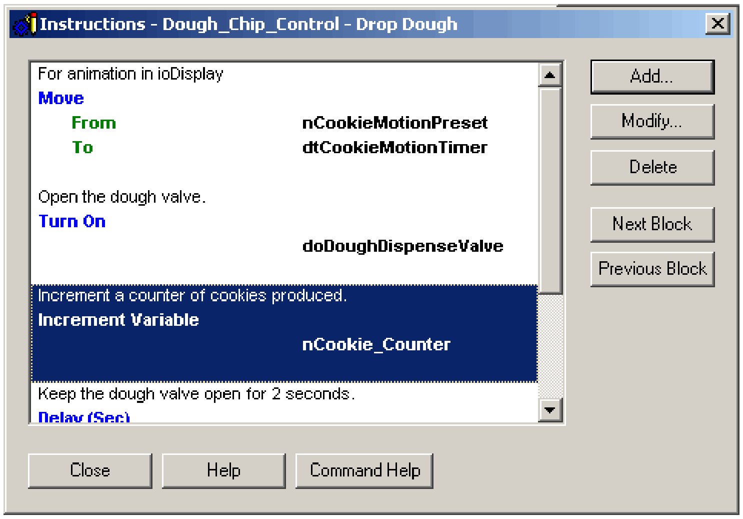
Initial Value: 0

Notice that the name (nCookie_Counter) and type (Integer 32) have already been filled in.

13. Add a description, if you wish. Leave the initial value at zero, which is the default. Then click OK to accept the data and close the dialog box.

14. In the Add Instruction dialog box, click OK.

The new instruction appears in the Instructions window for Drop Dough.



Instructions - Dough_Chip_Control - Drop Dough

```
For animation in ioDisplay
Move
     From                        nCookieMotionPreset
     To                          dtCookieMotionTimer

Open the dough valve.
Turn On
                                 doDoughDispenseValve

Increment a counter of cookies produced.
Increment Variable
                                 nCookie_Counter

Keep the dough valve open for 2 seconds.
Delay (Sec)
```

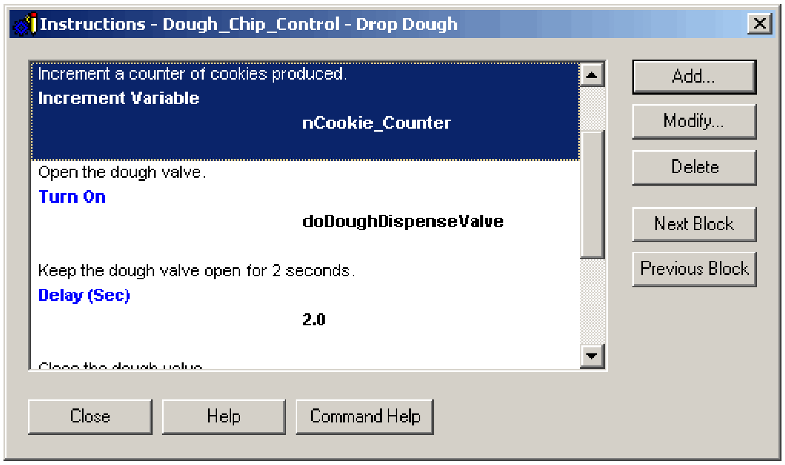
But maybe it makes more sense to start the counter at the beginning of the process, rather than in the middle, after some dough has already been dropped.

15. With the Increment Variable command highlighted, press the right mouse button and select Cut from the pop-up menu.

You can also use CTRL+X to cut. Cutting puts the instruction in the Windows Clipboard.

16. Now click Turn On to highlight it. Click the right mouse button and select Paste from the pop-up menu.

You can also use CTRL+V to paste. The Increment Variable command is pasted above the highlighted instruction, like this:



Instructions - Dough_Chip_Control - Drop Dough

```
Increment a counter of cookies produced.
Increment Variable
                                 nCookie_Counter

Open the dough valve.
Turn On
                                 doDoughDispenseValve

Keep the dough valve open for 2 seconds.
Delay (Sec)
                                 2.0

Close the dough valve
```

17. Click Close to return to the chart.

You've just added a cookie counter. On the Strategy Tree, open the Numeric Variables folder in the Variables folder. The new numeric variable is there.

18. Save the changes you've made by clicking the Save Strategy button on the toolbar. Click OK to save.

Now we're ready to download the strategy to a control engine. But first we have to tell ioControl that we have a control engine.

Configuring a Control Engine

Up to this point, we've been able to play around in ioControl without hardware. Now it's time to configure a control engine.

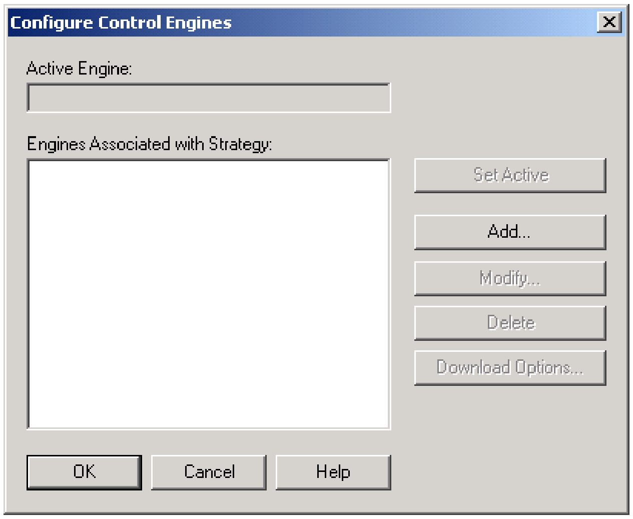
1. If you have an ioControl-compatible control engine and I/O unit you can use, make sure they are on the same network as your PC, and make sure you have loaded the most recent firmware.

For a list of compatible control engines and I/O units, see page 1-6. For instructions to load firmware, see Opto 22 form 1440, the *ioManager User's Guide*.

2. Turn the I/O unit off and then back on again.

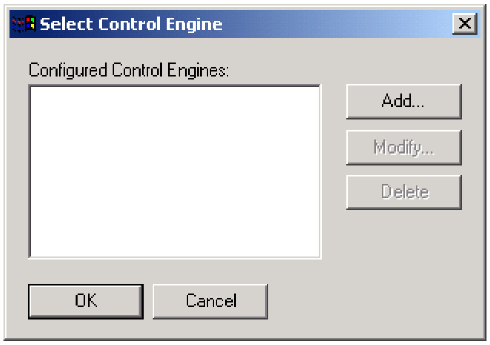
3. Double-click the Control Engines folder on the Strategy Tree, or click the Configure Control Engine button on the toolbar.

The Configure Control Engines dialog box appears.



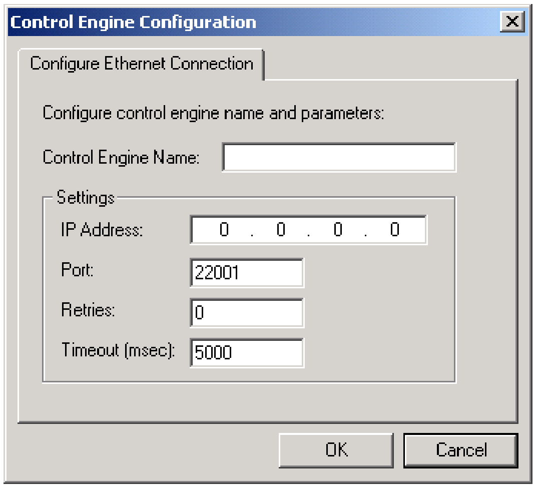
Since we haven't configured a control engine yet, there are no control engines in the list.

4. Click Add.



5. Click Add again to add a control engine.

The Control Engine Configuration dialog box appears.


Control Engine Configuration
Configure Ethernet Connection
Configure control engine name and parameters:
Settings
OK
Cancel

Control Engine Name:

IP Address: 0 . 0 . 0 . 0

Port: 22001

Retries: 0

Timeout (msec): 5000

6. Enter Cookie Controller as the control engine name.

The name can contain letters, numbers, spaces, and most other characters except colons and square brackets. Spaces cannot be used as first or last characters.

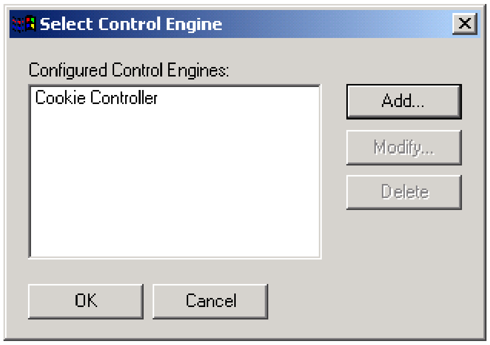
7. Enter the control engine's IP address.

On a hardware control engine such as a SNAP PAC or SNAP-LCE controller or a SNAP Ultimate brain, the IP address assigned to the device is usually written on a sticker on the side of the unit. If an IP address has not been assigned to the control engine, see Opto 22 form 1440, the *ioManager User's Guide,* for configuration instructions.

NOTE: In ioControl Professional, a second IP address field is available, so you can designate a secondary communication path to the control engine should the primary one fail. For more information, see "Using Ethernet Link Redundancy in ioControl" on page 5-6.

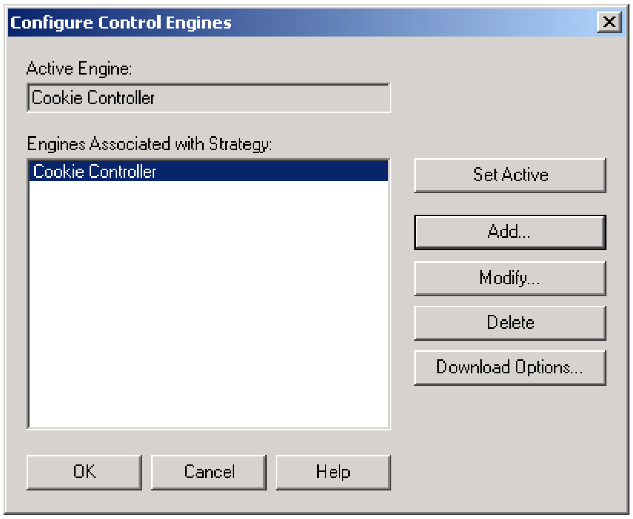
8. Make sure that you have not changed the values in the Port, Retries, and Timeout fields, and then click OK.

The newly configured control engine appears in the Select Control Engine dialog box.



9. Click the new Cookie Controller control engine to select it, and then click OK.

The new control engine appears in the Configure Control Engines dialog box.



Since you have only one configured control engine at this point, it is automatically set as the active control engine. If there were more than one control engine, you would have to select it and click Set Active to load it into the Active Engine field.

10. Click OK to close the Configure Control Engines dialog box.

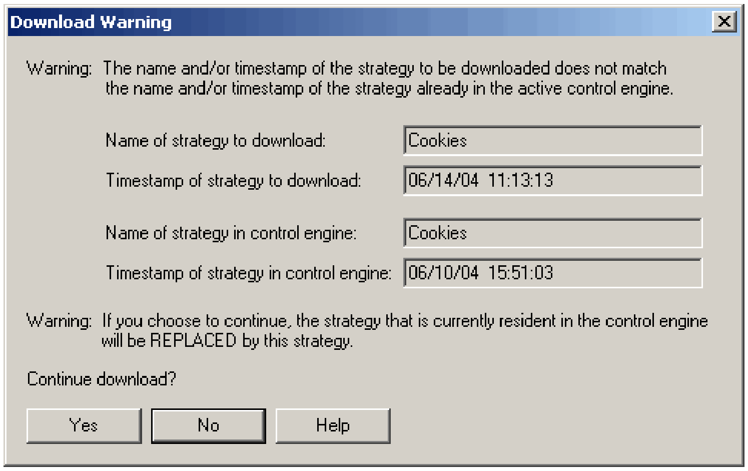
On the Strategy Tree, the new control engine appears as the first entry in the Control Engines folder.

Compiling the Strategy

The simplest way to compile a strategy is to enter Debug mode. The strategy is saved and compiled before changing modes.

1. Click the Debug Mode button on the toolbar (or select Debug from the Mode menu).
2. In the Save Strategy dialog box, click Yes to save the strategy.
3. If you see a Powerup Clear Expected message, click OK.

   You may see a Download Warning message like this one:



Download Warning

Warning: The name and/or timestamp of the strategy to be downloaded does not match the name and/or timestamp of the strategy already in the active control engine.

Name of strategy to download: Cookies

Timestamp of strategy to download: 06/14/04 11:13:13

Name of strategy in control engine: Cookies

Timestamp of strategy in control engine: 06/10/04 15:51:03

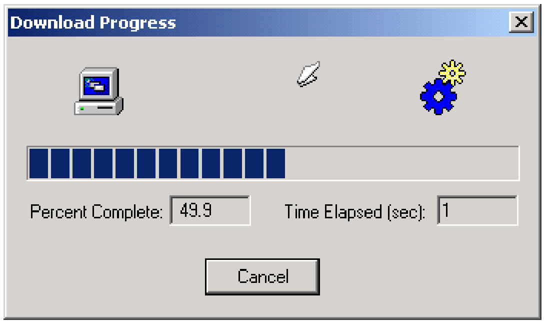
Warning: If you choose to continue, the strategy that is currently resident in the control engine will be REPLACED by this strategy.

Continue download?

Or you may see a message that the control engine's memory has been cleared.

4. Click Yes to proceed.

   Two additional dialog boxes appear briefly. The first displays the progress as the strategy is compiled. The second shows progress as the strategy is downloaded to the control engine.



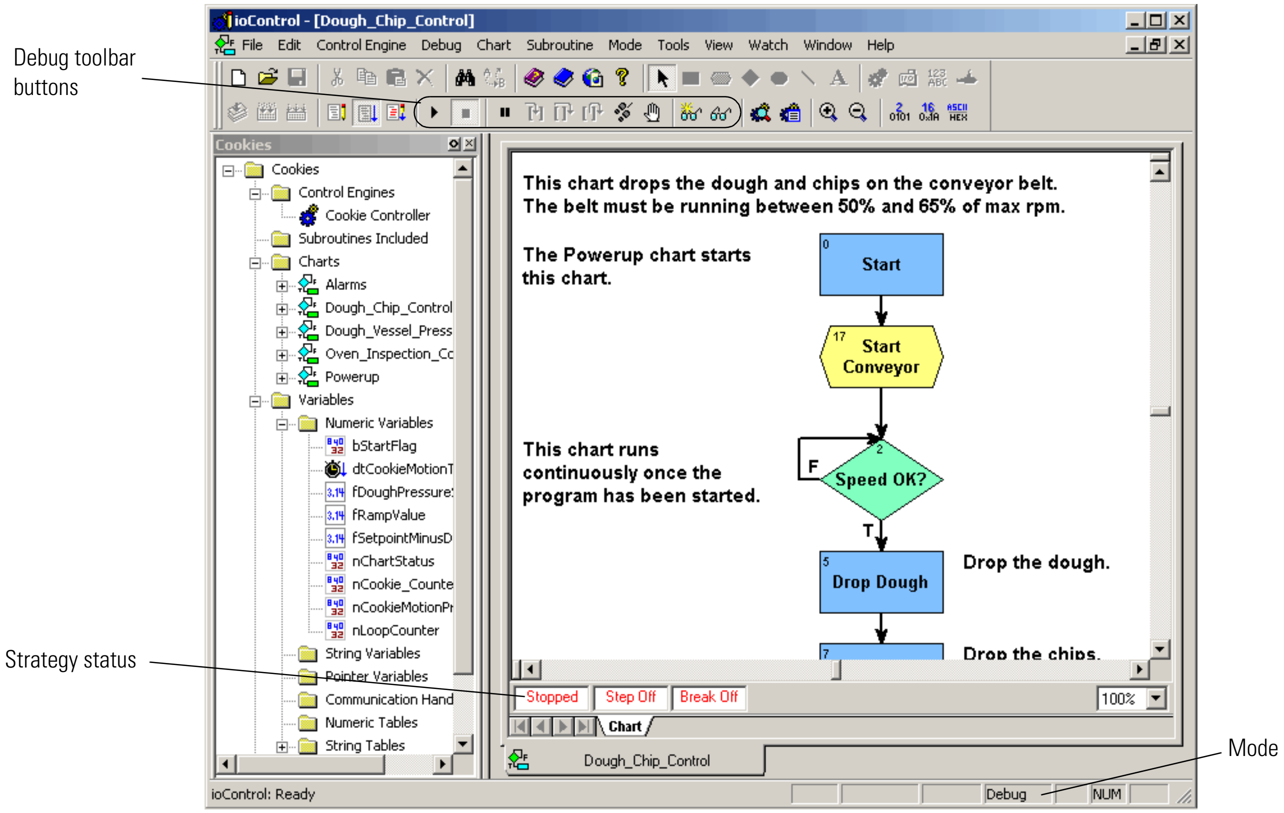
Download Progress

Percent Complete: 49.9

Time Elapsed (sec): 1

Assuming the strategy was compiled and downloaded successfully, you are now in Debug mode.

In the ioControl window, you’ll notice that the Debug toolbar is now available. The mode is shown at the bottom of the main window. The open chart window shows that the strategy is stopped, as shown in the following figure:



This chart drops the dough and chips on the conveyor belt. The belt must be running between 50% and 65% of max rpm.

The Powerup chart starts this chart.

This chart runs continuously once the program has been started.

Drop the dough.

Drop the chips.

Running the Strategy

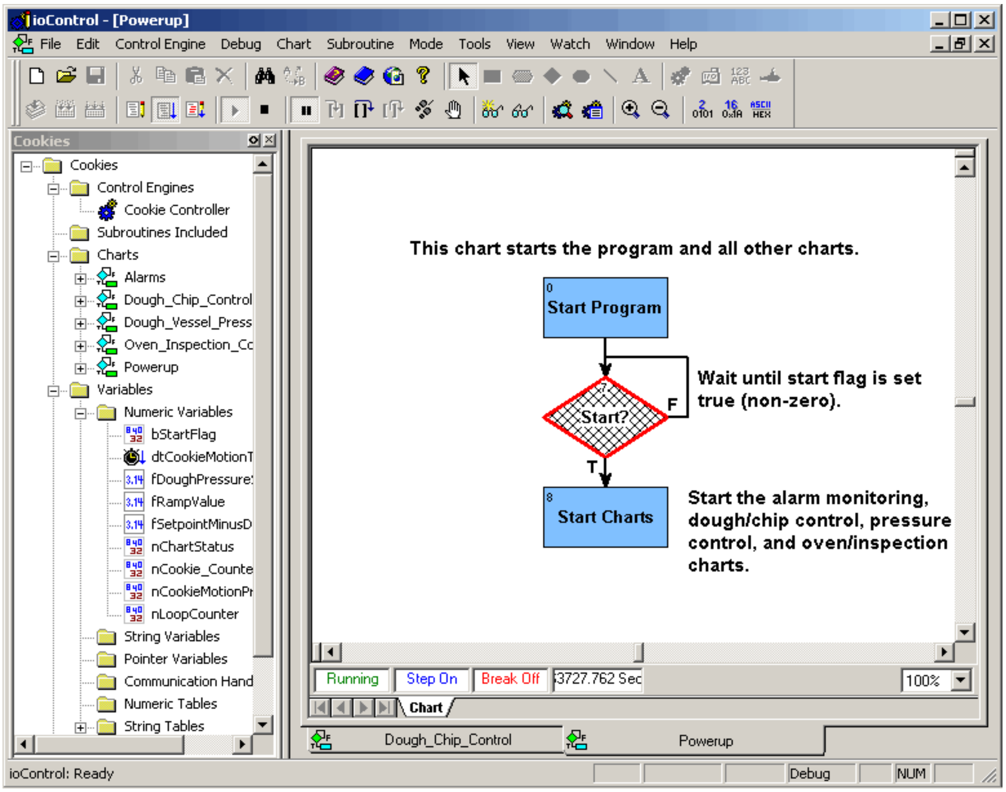
In Debug mode, we’re going to run our strategy and examine it. We’ll see how the strategy run affects variables, how the command blocks are executed, and so on. The first chart to run in any strategy is the Powerup chart, so we’ll look at it first.

1. Double-click the Powerup chart on the Strategy Tree. When it opens, notice that it says Stopped at the bottom left.

2. Click the Run button ▶.

   At the bottom of the chart window, the word Stopped changes to Running. Let’s try pausing the program to see where we are.

3. Click the Pause button ⏸.



This chart starts the program and all other charts.

Wait until start flag is set true (non-zero).

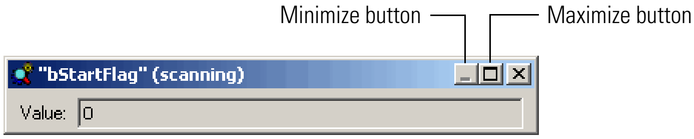
Start the alarm monitoring, dough/chip control, pressure control, and oven/inspection charts.

The hatch marks on the Start? block indicate that this command block was about to be executed when we clicked Pause. Apparently the program isn't getting past this block. Notice that the False exit routes right back to the top of the Start? block, while the True exit moves on to Start Charts. We can see that if the start flag had been true (non-zero), the program would have gone right into the Start Charts block. Since we didn't get that far, the start flag must be zero.

And in fact it is. We planned it that way because we wanted someone (for example, a factory operator) to start the process intentionally. We can simulate this ourselves in ioControl by manually setting the flag to a non-zero value.

4. Double-click the bStartFlag variable on the Strategy Tree.

A little box appears.

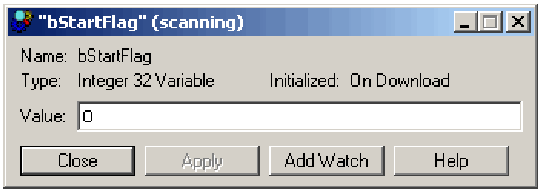


"bStartFlag" (scanning)

Value: 0

In this dialog box you can view the variable value, but you cannot change it unless you maximize the dialog box.

5. Click the Maximize button.



Name: bStartFlag
Type: Integer 32 Variable   Initialized: On Download
Value: 0

This dialog box displays current information on the variable bStartFlag. You can see that the variable is initialized to zero on strategy download.

6. Highlight the value zero in the dialog box and type `1` to replace it.

   The field turns purple, indicating a change has been made but not implemented.

7. Click Apply to implement the change. Click the Minimize button in the bStartFlag View Variable dialog box and move it out of the way. Click the Powerup tab to make the Powerup chart the active window.

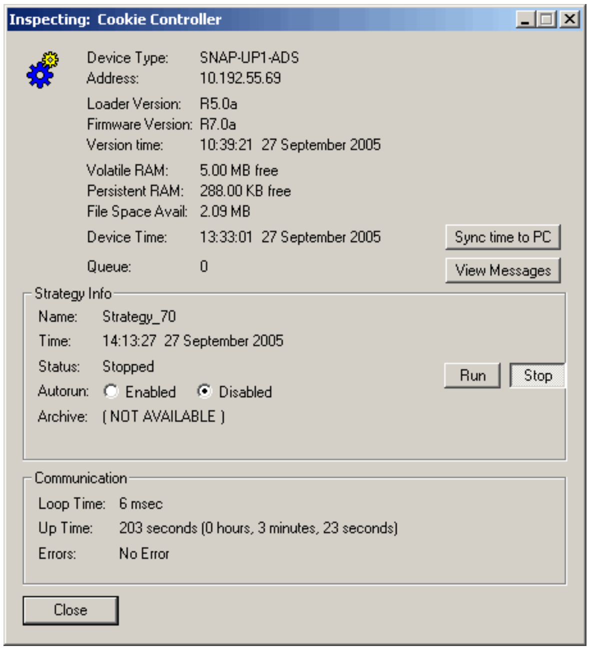
Inspecting Messages

1. Look for a light-blue INFO box in the status bar at the bottom of the window.

   The status bar in the main ioControl window tells you when an information, warning, or error message has been placed in the message queue. Messages can help in troubleshooting your strategy. In this example an Information message is in the message queue.

2. Click the INFO box.

   The dialog box that appears (illustrated at right) shows you information about the control engine.

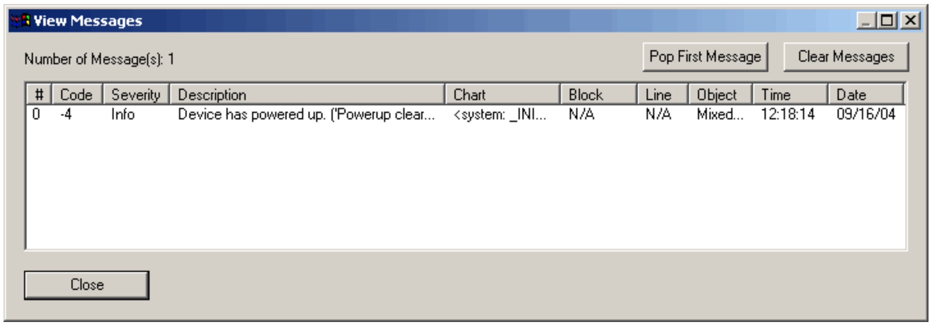


Device Type: SNAP-UP1-ADS
Address: 10.192.55.69
Loader Version: R5.0a
Firmware Version: R7.0a
Version time: 10:39:21 27 September 2005
Volatile RAM: 5.00 MB free
Persistent RAM: 288.00 KB free
File Space Avail: 2.09 MB
Device Time: 13:33:01 27 September 2005
Queue: 0

Name: Strategy_70
Time: 14:13:27 27 September 2005
Status: Stopped
Autorun: ○ Enabled ● Disabled
Archive: (NOT AVAILABLE)

Loop Time: 6 msec
Up Time: 203 seconds (0 hours, 3 minutes, 23 seconds)
Errors: No Error

3. Click View Messages to see the information message.



Number of Message(s): 1

#	Code	Severity	Description	Chart	Block	Line	Object	Time	Date
0	-4	Info	Device has powered up. ('Powerup clear...	<system: _INI...	N/A	N/A	Mixed...	12:18:14	09/16/04

This message tells you that the I/O unit recently lost power but is now on.

4. Close both dialog boxes to return to the Powerup chart.

Stepping Through the Chart

Now let's step through the chart to see what's happening.

1. Click inside the Powerup chart, and then click the Step Over button .

The hatch marks move from Start? to Start Charts. We've just moved (stepped) to the next block.

2. Click the button again.

ioControl executes the Start Charts block and steps to the next command block. Since there are no more blocks in this chart, we are actually exiting this chart and moving on to new instructions in another chart. The Powerup chart has stopped, and you can see the word Stopped at the bottom left of the chart.

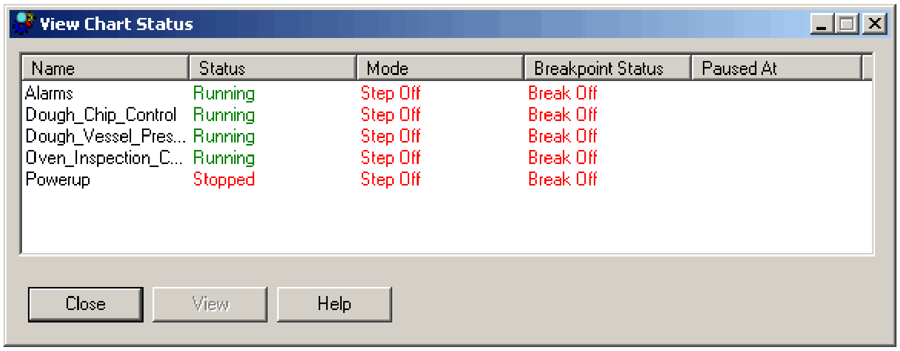
In the View Variable dialog box, the bStartFlag value reverts to zero, because the Start Charts block set the flag back to zero.

3. Close the bStartFlag View Variable dialog box. In the Powerup chart window, click the Pause button to turn stepping off.

Then close the Powerup chart.

4. Double-click the Charts folder on the Strategy Tree.

The View Chart Status dialog box appears.



Name	Status	Mode	Breakpoint Status	Paused At
Alarms	Running	Step Off	Break Off	
Dough_Chip_Control	Running	Step Off	Break Off	
Dough_Vessel_Pres...	Running	Step Off	Break Off	
Oven_Inspection_C...	Running	Step Off	Break Off	
Powerup	Stopped	Step Off	Break Off	

This dialog box shows us all the charts in our strategy. As you can see, four of the charts are running and one (Powerup) is stopped.

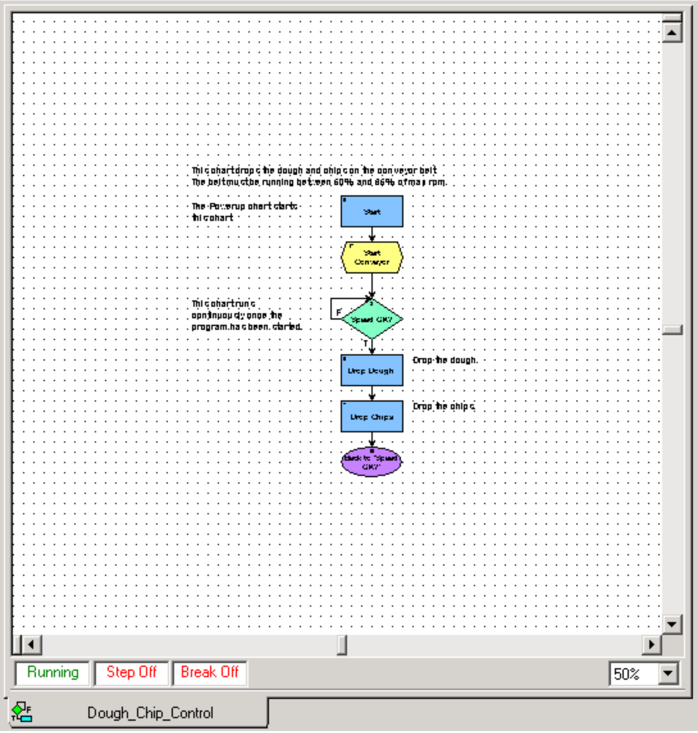
Powerup is stopped because it has already done its job. Let's check the running charts.

5. Close the View Chart Status dialog box and click inside the Dough_Chip_Control chart again.

6. Click the Zoom Out button on the toolbar to view the window at a smaller percentage.

There are several other ways to zoom out, too. You can right-click on the chart and select Zoom from the pop-up menu. You can select Zoom Out from the View menu. You can press the + or - keys on the keyboard. If your mouse has a wheel, you can also hold down the CTRL key and move the mouse wheel up or down to change the zoom.

After zooming out, the chart looks something like this:



This chart drops the dough and chips on the conveyor belt. The belt must be running between 60% and 86% of max rpm.

The Powerup chart starts this chart

This chart runs continuously once the program has been started.

Drop the dough.

Drop the chips.

You can zoom back in if you wish, using the Zoom In button on the toolbar or one of the other methods.

The chart's status bar indicates that it is running, but we can't see much happening.

7. Click the Pause button ⏸ .

One of the command blocks appears with hatch marks.

8. Now click the Step Over button .

The next command block is hatched.

9. Continue clicking and watch how the program proceeds from Speed OK? to Drop Dough to Drop Chips to Back to 'Speed OK?' and back up to Speed OK?

A pulsating green border appears around a block when the commands inside the block are being executed. While you are stepping through, and anytime the Pause button is clicked, the chart status indicates Step On.

10. Click the Pause button again to run the strategy at full speed.

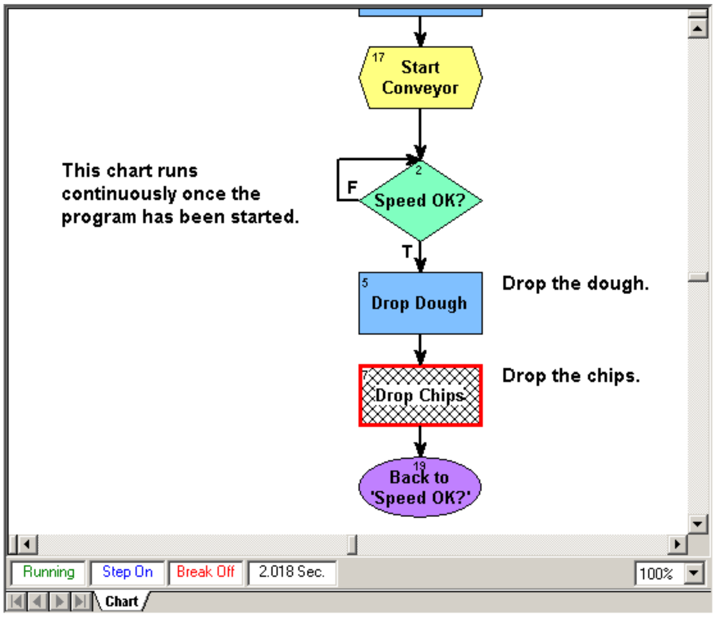
Step Off now appears in the chart status bar.

Auto Stepping

The final stepping option is auto stepping. You can select this option whether or not the chart is currently paused.

1. Click the Auto Step Chart button and watch the program move from block to block.

   At one time or another, your chart looks like this:



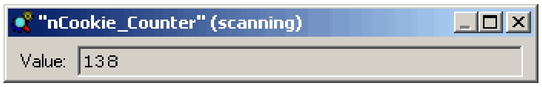
   Notice that in the ioControl toolbar, the Run and Auto Step Chart buttons are depressed. The chart status bar shows us the chart is running and in Step Auto mode. The time indicator to the right of Break Off shows the time it took for the most recent block to execute.

2. Click the Auto Step Chart button again to end auto stepping.

   Step Off appears, indicating the program is again running without interruption.

   Now let's see how many cookies we've produced to this point.

3. On the Strategy Tree, double-click the numeric variable nCookie_Counter.



"nCookie_Counter" (scanning)

Value: 138

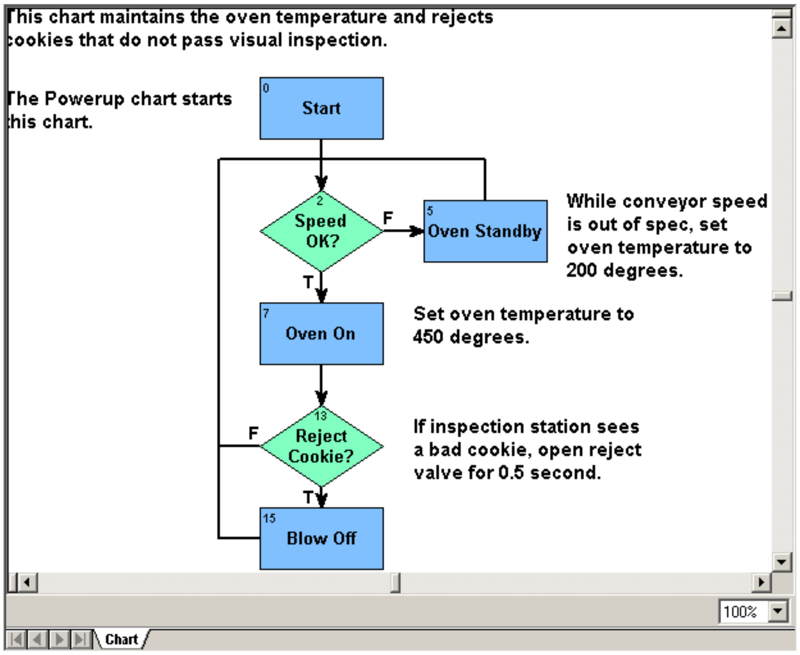
   The Value field should increase every time a cookie is produced, adding to the total number of cookies produced since the strategy run began. The nCookie_Counter above shows this figure as 138. (Yours may be different.)

   But nCookie_Counter tells us the total number of cookies put on the conveyor belt, without considering that some of them may be rejected by the inspection station. We need to subtract the number of bad cookies so that nCookie_Counter keeps track of the number of cookies sent out the door, not just sent to the oven.

4. Close the nCookie_Counter View Variable window and click the Configure mode button on the toolbar.

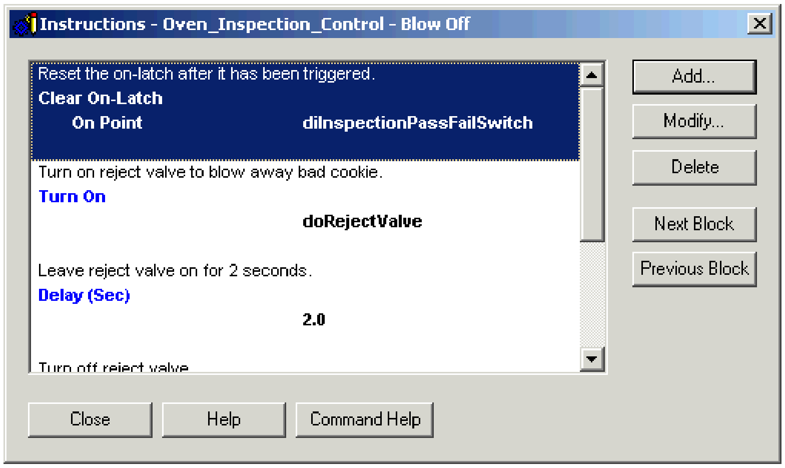
5. Double-click the Oven_Inspection_Control chart on the Strategy Tree to open it.

The chart window looks something like this:


This chart maintains the oven temperature and rejects cookies that do not pass visual inspection.
The Powerup chart starts this chart.
0
Start
2
Speed OK?
F
5
Oven Standby
While conveyor speed is out of spec, set oven temperature to 200 degrees.
T
7
Oven On
Set oven temperature to 450 degrees.
13
Reject Cookie?
F
If inspection station sees a bad cookie, open reject valve for 0.5 second.
T
15
Blow Off
100%
Chart

Near the bottom of the chart, the Reject Cookie? block determines whether a bad cookie has been found. If one has, the strategy moves to the next block, Blow Off, which is where the bad cookie gets blown off the conveyor. When that happens, we want to decrement the nCookie_Counter variable.

6. Double-click the Blow Off block to open its instructions window:



Instructions - Oven_Inspection_Control - Blow Off

Reset the on-latch after it has been triggered.
Clear On-Latch
On Point dilnspectionPassFailSwitch

Turn on reject valve to blow away bad cookie.
Turn On
doRejectValve

Leave reject valve on for 2 seconds.
Delay (Sec)
2.0

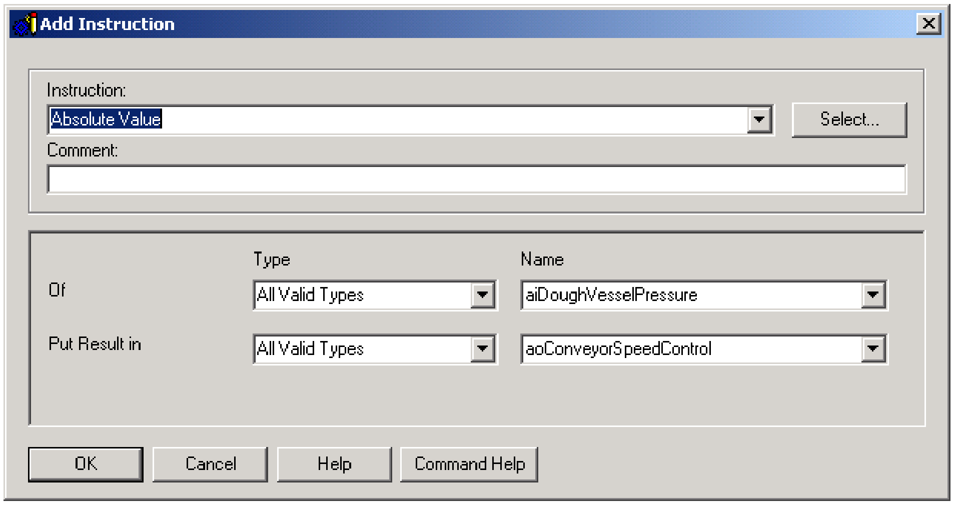
Turn off reject valve

This command block is executed only when a bad cookie has been found. This block first resets the on-latch triggered by the bad cookie, so that the next cookie won't be marked bad, too. The block then turns on the reject valve. The valve stays on for two seconds before being shut off. Let's decrement the counter after the cookie is gone and the valve is shut.

7. Scroll down and click on the open spot, below the other instructions.

The highlighted line marks the position of the next command to be added.

8. Click Add.



Add Instruction

Instruction: Absolute Value

Comment:

	Type	Name
Of	All Valid Types	aiDoughVesselPressure
Put Result in	All Valid Types	aoConveyorSpeedControl

You can use ioControl without a mouse, and to demonstrate how, we'll use only the keyboard to enter data in this dialog box.

9. Type `dec` in the Instruction field.

The Decrement Variable command appears, since it's the first command that starts with that text pattern. This is the command we want to use.

10. Press TAB twice to move to the Comment field. Type in a comment.

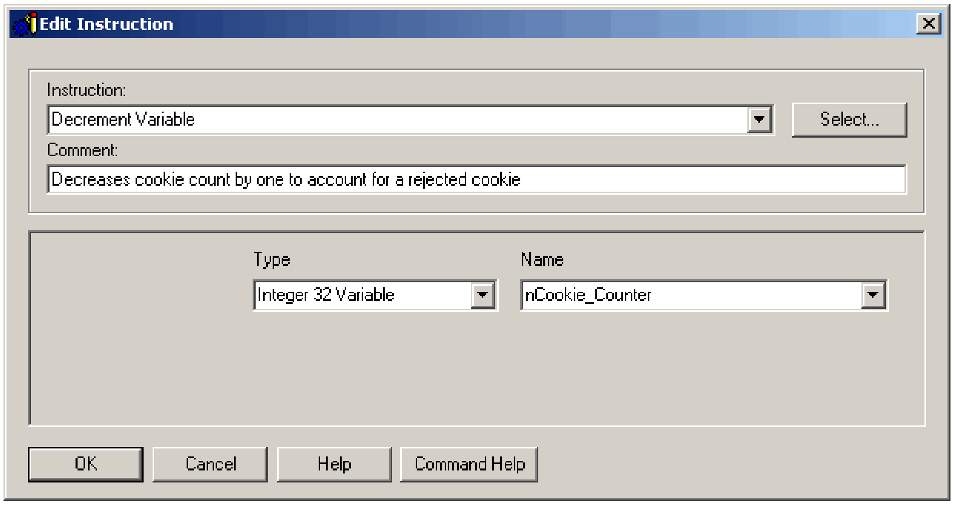
11. Press TAB again to move to the Type field. Press the down arrow on your keyboard twice to select Integer 32 Variable.

12. Press TAB again to advance to the Name field, and then press the down arrow until you see nCookie_Counter.

13. Press TAB again and notice that an outline appears on the OK button.

An outlined button means that pressing the space bar or ENTER is equivalent to clicking the button.

Now the dialog box looks like this:



Edit Instruction

Instruction: Decrement Variable

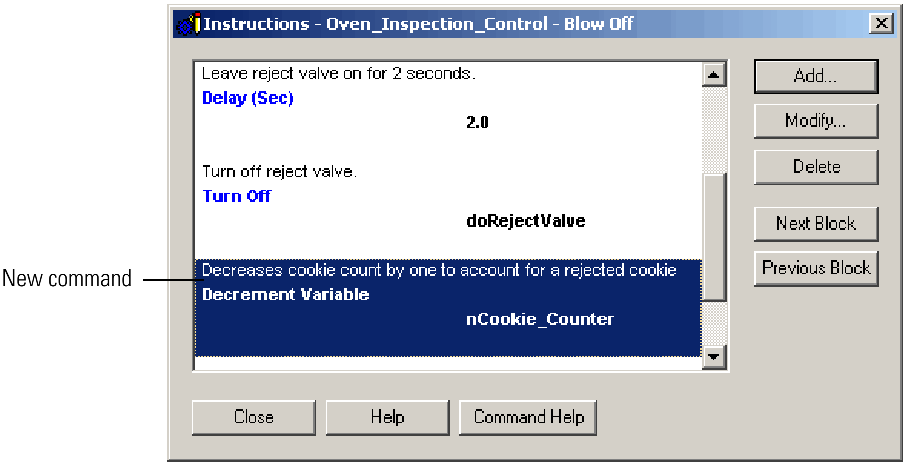
Comment: Decreases cookie count by one to account for a rejected cookie

Type: Integer 32 Variable

Name: nCookie_Counter

14. Press ENTER.

The dialog box closes and the new command appears in the Instructions window.



Instructions - Oven_Inspection_Control - Blow Off

Leave reject valve on for 2 seconds.
Delay (Sec)
2.0

Turn off reject valve.
Turn Off
doRejectValve

Decreases cookie count by one to account for a rejected cookie
Decrement Variable
nCookie_Counter

15. Click Close to return to the Oven_Inspection_Control chart.

Compiling and Downloading the Change

Now we'll compile and download the modified strategy.

1. Click the Debug mode button on the toolbar.

   A message box appears, warning you that changes have been detected and asking if you want to save them before downloading

2. Click Yes to continue.

   Another warning notes that the strategy name or timestamp differs from that in the control engine and asks if you want to continue.

3. Click Yes.

4. On the toolbar, click the Run button . In the Strategy Tree, doubleclick the bStartFlag variable. Maximize the dialog box, change the value to 1, and click Apply. Close the dialog box.

5. Click inside the Oven_Inspection_Control chart to make it the active chart, and then click the Auto Step Chart button.

   You see three blocks being processed: Speed OK?, Oven On, and Reject Cookie? The strategy doesn't move into the Blow Off block. That's because an inspector has to flag bad cookies, but we don't have an inspector right now. So we'll simulate what would happen by tripping the digital input that flags a bad cookie.

6. In the Oven_Inspection_Control chart, click the Auto Step Chart button again to stop auto stepping.

   Step Auto changes to Step Off in the status bar.

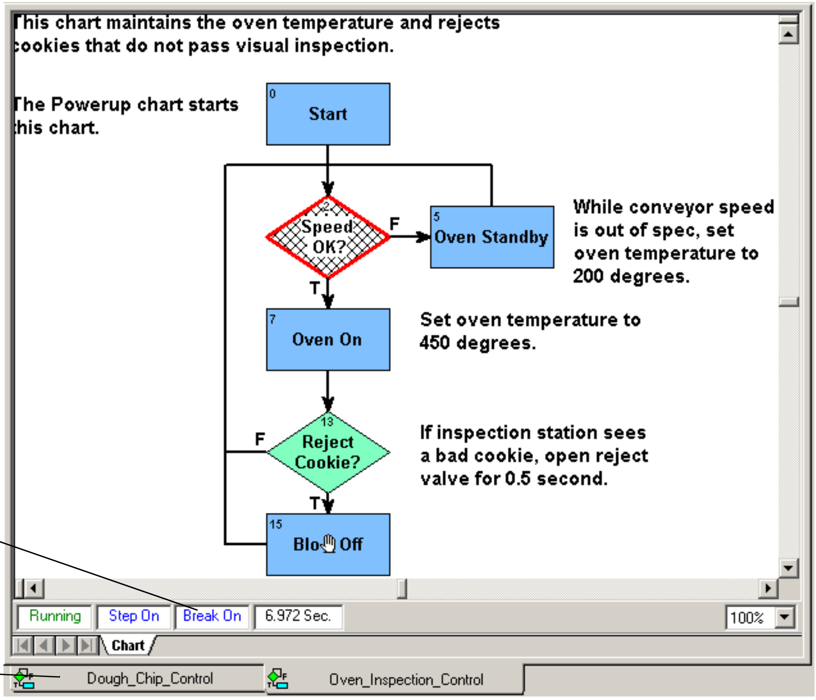
7. Click the Pause button.

   Step Off changes to Step On.

8. Click the Breakpoint Tool button and click once on the Blow Off block.

   A breakpoint hand appears on the block.

9. Click the right mouse button or press ESC to release the tool. Notice that Break On now appears in the chart status bar.



This chart maintains the oven temperature and rejects cookies that do not pass visual inspection.

The Powerup chart starts this chart.

While conveyor speed is out of spec, set oven temperature to 200 degrees.

Set oven temperature to 450 degrees.

If inspection station sees a bad cookie, open reject valve for 0.5 second.

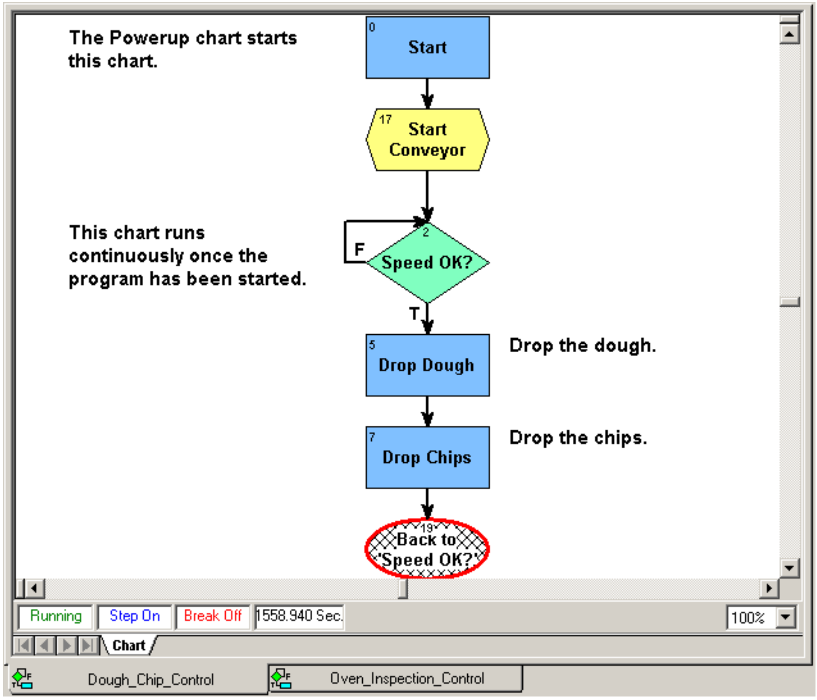
Break On

Dough_Chip_Control tab

10. Click the Dough_Chip_Control tab at the bottom of the ioControl main window.

The chart appears.

11. Click the Pause button to pause the chart.



The Powerup chart starts this chart.

This chart runs continuously once the program has been started.

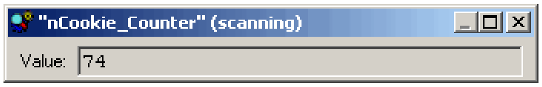
Drop the dough.

Drop the chips.

Using a Watch Window

To see clearly what we're doing, we'll create a watch window to monitor cookie production.

1. In the Strategy Tree, double-click nCookie_Counter.

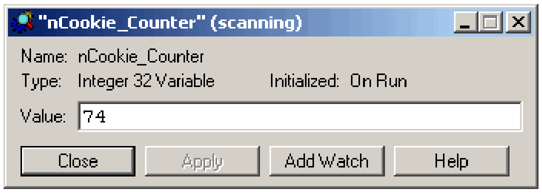


"nCookie_Counter" (scanning)

Value: 74

The value is frozen at some number, such as the 74 shown above. Since the counter is no longer increasing, we can see that cookie production has temporarily stopped.

2. Maximize the dialog box.



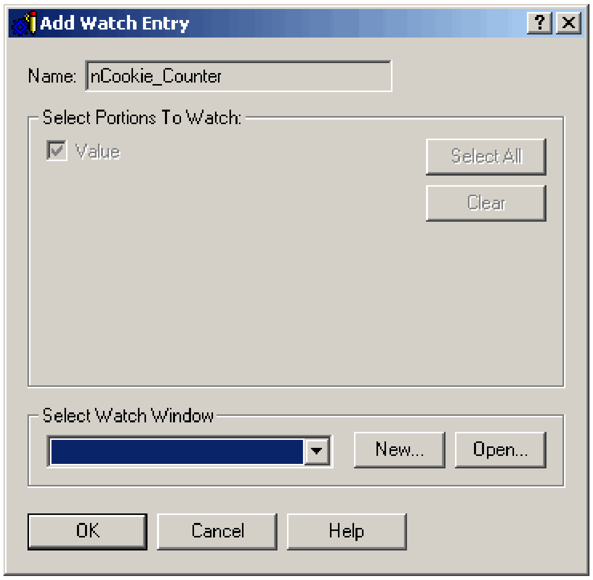
"nCookie_Counter" (scanning)

Name: nCookie_Counter

Type: Integer 32 Variable Initialized: On Run

Value: 74

3. Click Add Watch.



Add Watch Entry

Name: nCookie_Counter

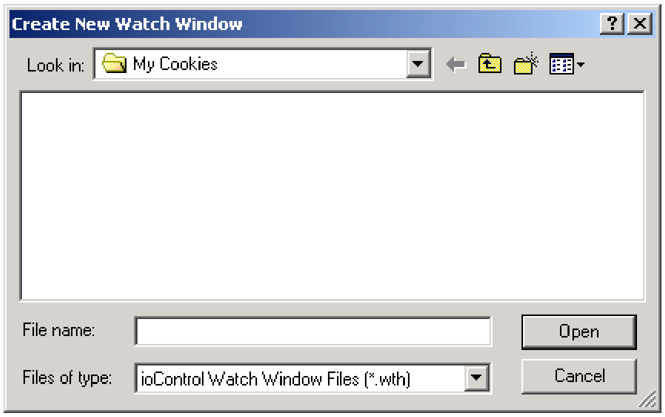
Select Portions To Watch:

☑ Value

Select Watch Window

Since there is no watch window available to select, we'll create one.

4. Click New.



Create New Watch Window

Look in: My Cookies

File name:

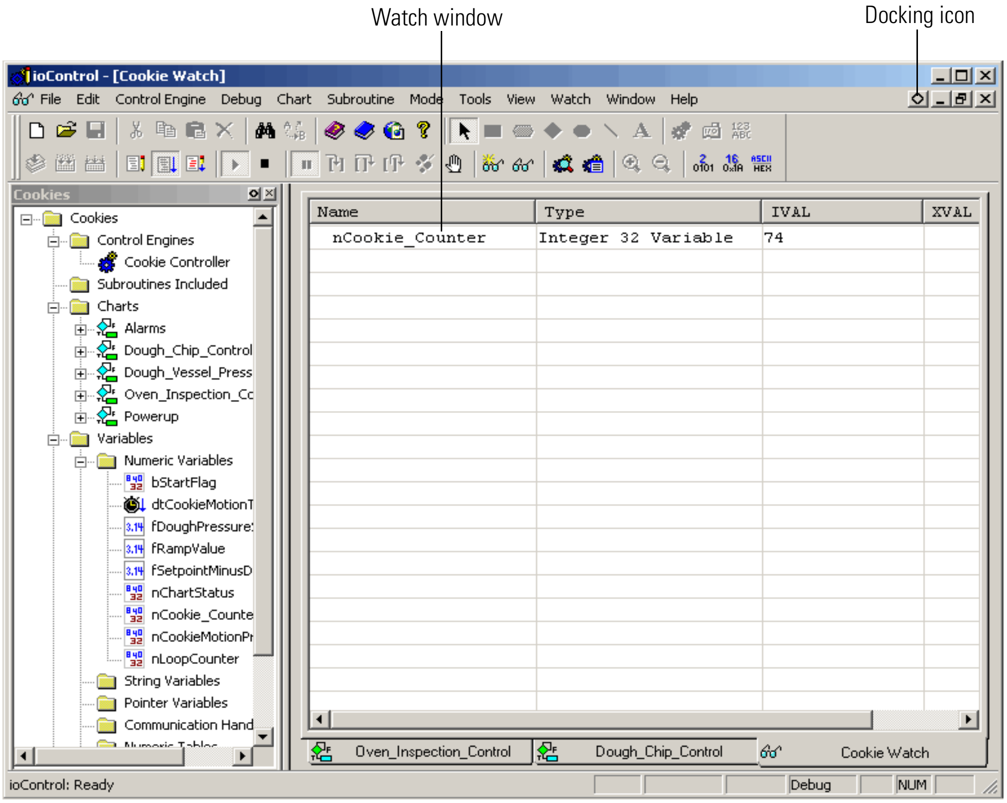
Files of type: ioControl Watch Window Files (*.wth)

5. Make sure the My Cookies directory appears in the Look in field. Type a name for the watch window in the File name field. Then click Open.

The watch window name appears in the Add Watch dialog box, and the new watch window appears behind it.

6. In the Add Watch Entry dialog box, click OK. Close the nCookie_Counter view variable dialog box.

The new watch window looks something like this:



Watch window

Docking icon

ioControl - [Cookie Watch]

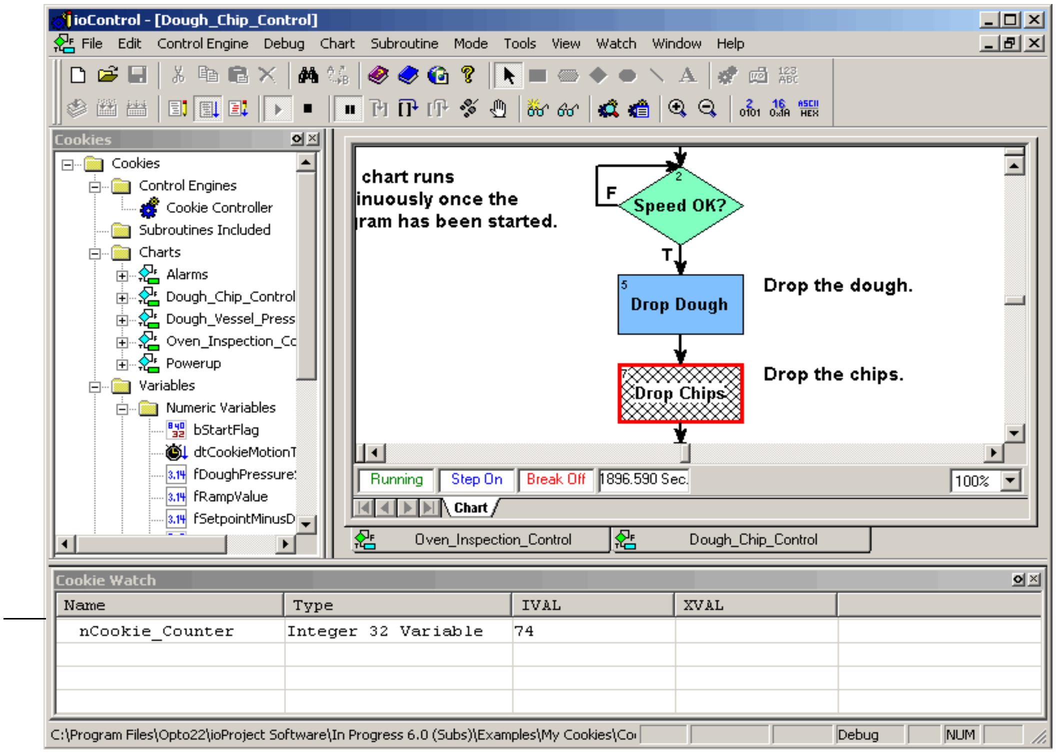
Name	Type	IVAL	XVAL
nCookie_Counter	Integer 32 Variable	74	

ioControl: Ready

Since we want to be able to see chart windows as well as the watch window, let's dock the watch window at the bottom.

7. In the watch window, click the docking button in the upper-right corner.

The watch window moves to the bottom of the main window.



Name	Type	IVAL	XVAL
nCookie_Counter	Integer 32 Variable	74	

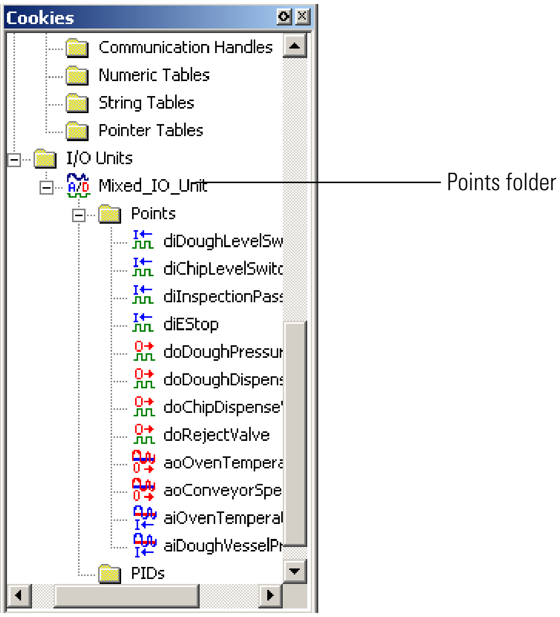
Docked watch window

Now we'll trip the latch that signals a bad cookie.

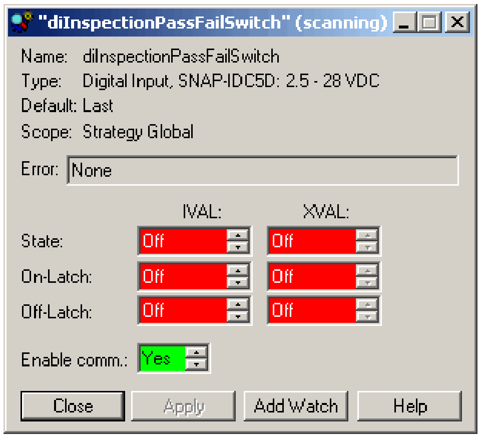
8. On the Strategy Tree, under the I/O Units folder at the bottom of the window, expand Mixed_IO_Unit by clicking the plus sign at the left of the name.

You see a folder named Points.

9. Expand the Points folder to display the digital I/O points configured for this I/O unit.



10. Double-click diInspectionPassFailSwitch. In the minimized dialog box, click the maximize button.



Name: diInspectionPassFailSwitch
Type: Digital Input, SNAP-IDC5D: 2.5 - 28 VDC
Default: Last
Scope: Strategy Global

Error: None

	IVAL:	XVAL:
State:	Off	Off
On-Latch:	Off	Off
Off-Latch:	Off	Off

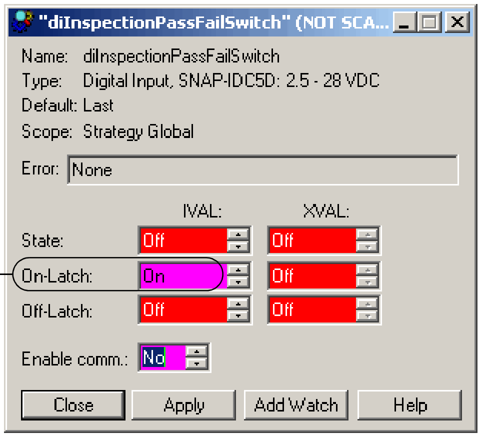
Enable comm.: Yes

NOTE: Don't worry if the red error message "I/O unit not configured" or XVAL values appear in this dialog box. This occurs because the strategy is configured for sample hardware that probably doesn't correspond to your actual I/O units and modules. ioControl can't locate this hardware. That's okay for this example.

When an inspection finds a bad cookie, the on-latch attached to the I/O point diInspectionPassFailSwitch is supposed to go on. We're going to trip it manually.

11. Click one of the arrows in the On-Latch IVAL field to change it to On. Also make sure that the Enable comm field says No.

The dialog box should look like this:



"diInspectionPassFailSwitch" (NOT SCA...

Name: diInspectionPassFailSwitch
Type: Digital Input, SNAP-IDC5D: 2.5 - 28 VDC
Default: Last
Scope: Strategy Global

Error: None

	IVAL:	XVAL:
State:	Off	Off
On-Latch:	On	Off
Off-Latch:	Off	Off

Enable comm.: No

On-latch On

12. Click Apply.

    The On-Latch IVAL field turns green after a second or two, indicating the latch is on.

13. Click Add Watch.

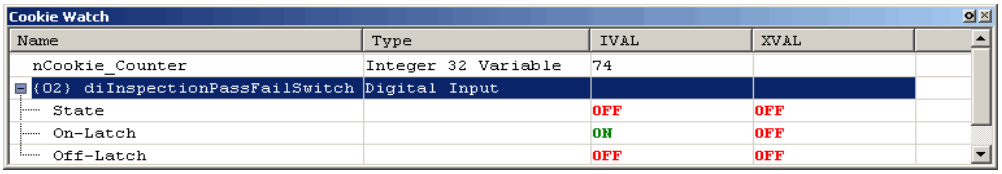
    We'll add the variable to our watch window, so we can see what happens.

14. In the Add Watch dialog box, leave all portions checked. Click OK to add the variable to the Cookie Watch window. Close the view point dialog box.

15. In the Cookie Watch window, click the plus sign next to (02) diInspectionPassFailSwitch.

    Your screen may show only part of the words.

16. Move your cursor over the right side of the Name column until the cursor changes shape. Then click and drag the column to make it wider, until you can see all the words.

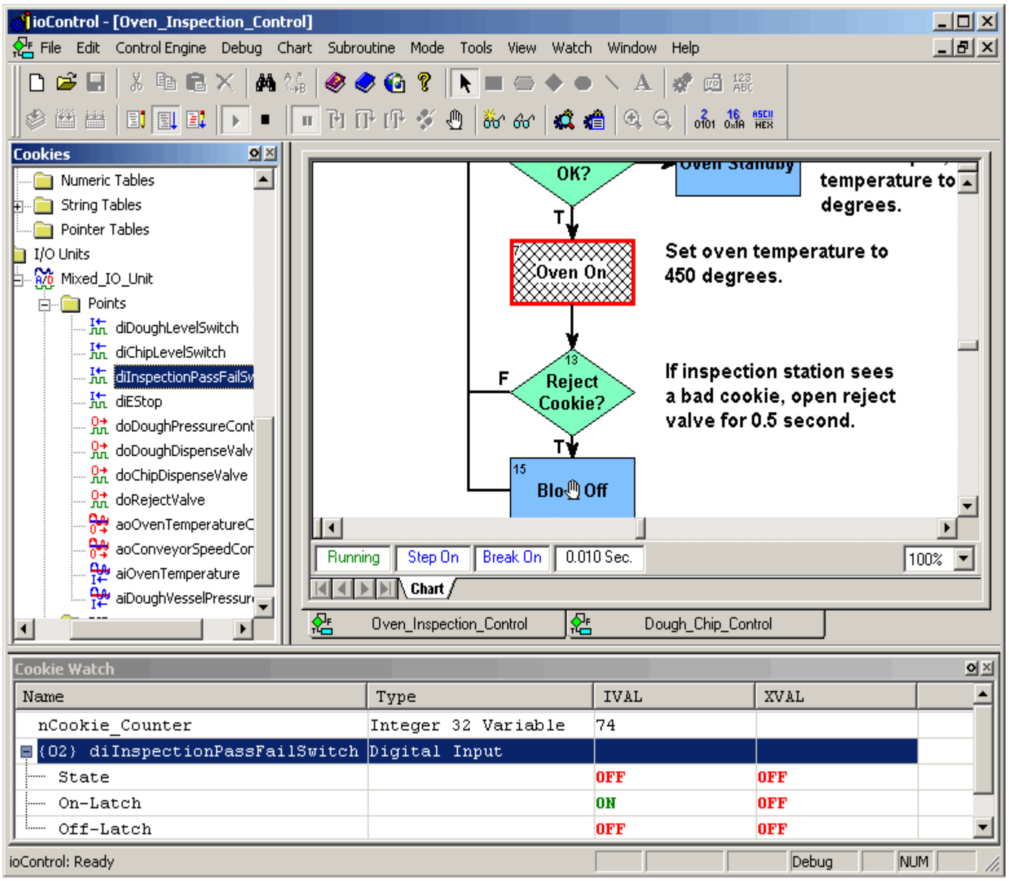


Cookie Watch

Name	Type	IVAL	XVAL
nCookie_Counter	Integer 32 Variable	74	
⊟ (02) diInspectionPassFailSwitch	Digital Input		
State		OFF	OFF
On-Latch		ON	OFF
Off-Latch		OFF	OFF

17. Click in the Oven_Inspection_Control chart to make it active, and move the scroll bars until you can see the Blow Off block at the bottom.

Your window now looks something like this:



temperature to degrees.

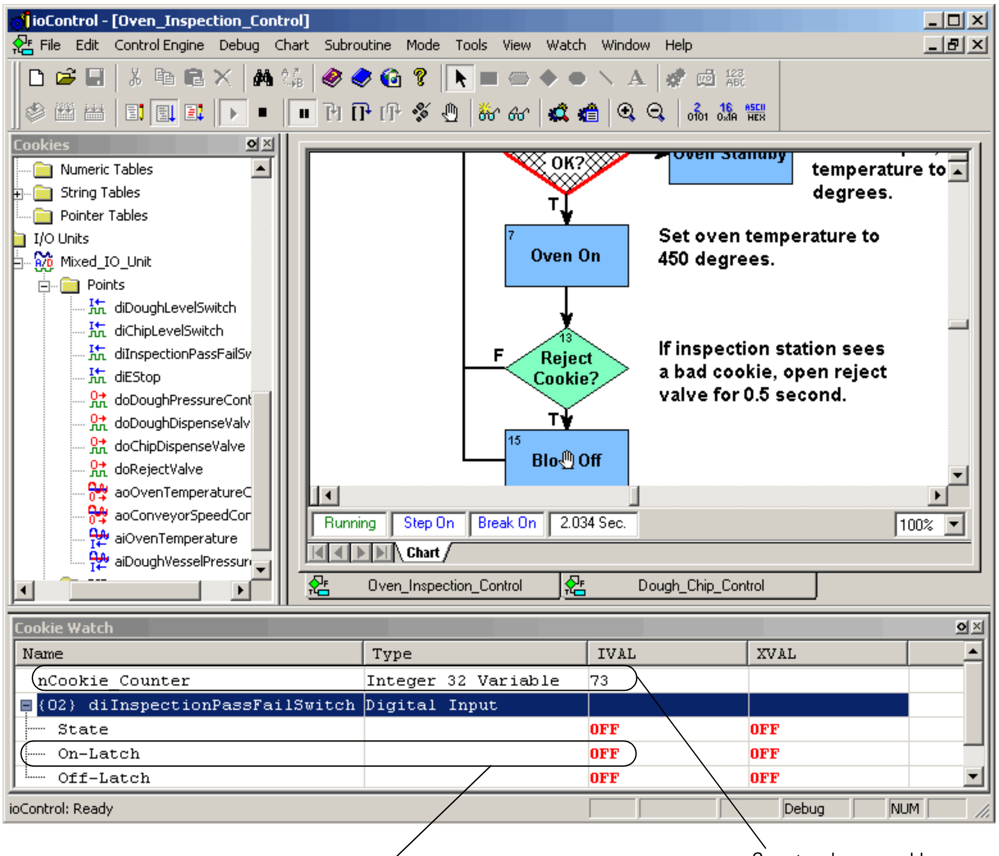
Set oven temperature to 450 degrees.

If inspection station sees a bad cookie, open reject valve for 0.5 second.

Name	Type	IVAL	XVAL
nCookie_Counter	Integer 32 Variable	74	
⊟ (02) diInspectionPassFailSwitch	Digital Input		
State		OFF	OFF
On-Latch		ON	OFF
Off-Latch		OFF	OFF

18. Click the Step Over button as many times as it takes to get the hatch marks to the Blow Off block. Now watch the nCookie_Counter IVAL value in the watch window as you click the button again.

A bad cookie was found, so the counter was decreased by one. At the same time, the on-latch was reset to Off, as you can also see in the watch window.



Name	Type	IVAL	XVAL
nCookie_Counter	Integer 32 Variable	73	
{02} diInspectionPassFailSwitch	Digital Input		
State		OFF	OFF
On-Latch		OFF	OFF
Off-Latch		OFF	OFF

On-latch reset to Off

Counter decreased by one

19. Click the Auto Step Chart button to go back to auto stepping.

The counter does not decrease again, because the on-latch is no longer set. But the counter won't increase until we start the Dough_Chip_Control chart again.

20. Click the Dough_Chip_Control chart tab. Click the Pause button to unpause the chart. Verify that Step On changes to Step Off in the chart status bar.

The watch window shows the nCookie_Counter value going up again.

Closing the Strategy and Exiting

Before we finish this tutorial, you may want to explore the sample strategy on your own. You can double-click items in the Strategy Tree or open up other charts to see what they do. You can also double-click command blocks to see what they contain.

1. When you're ready to stop, click the Stop button in the toolbar.

   This action prevents the strategy from running continuously unattended on the control engine.

2. To close the strategy, select File→Close Strategy; or select File→Exit to quit ioControl. If dialog boxes ask whether you want to remove breakpoints and take charts out of stepping mode, click Yes.

What's Next?

Your introduction to ioControl is now complete. Using the sample Cookies strategy, you have learned how to:

- Open, save, and close a strategy
- Work with the Strategy Tree
- Work with charts and add commands in blocks
- Configure a control engine
- Compile, run, step through, and add breakpoints to a strategy
- Make an online change
- Use a watch window to monitor variables and I/O points.

The rest of this book expands on the knowledge you've just acquired. Now may be a good time to look at the table of contents or thumb through the book and familiarize yourself with its contents. Some sections you may want to read; others you'll probably just refer to as needed.

CHAPTER 3

What Is ioControl?

Introduction

The tutorial in Chapter 2 introduced you to ioControl without explaining much about it. In this chapter we'll learn more about ioControl and see its main windows and toolbars.

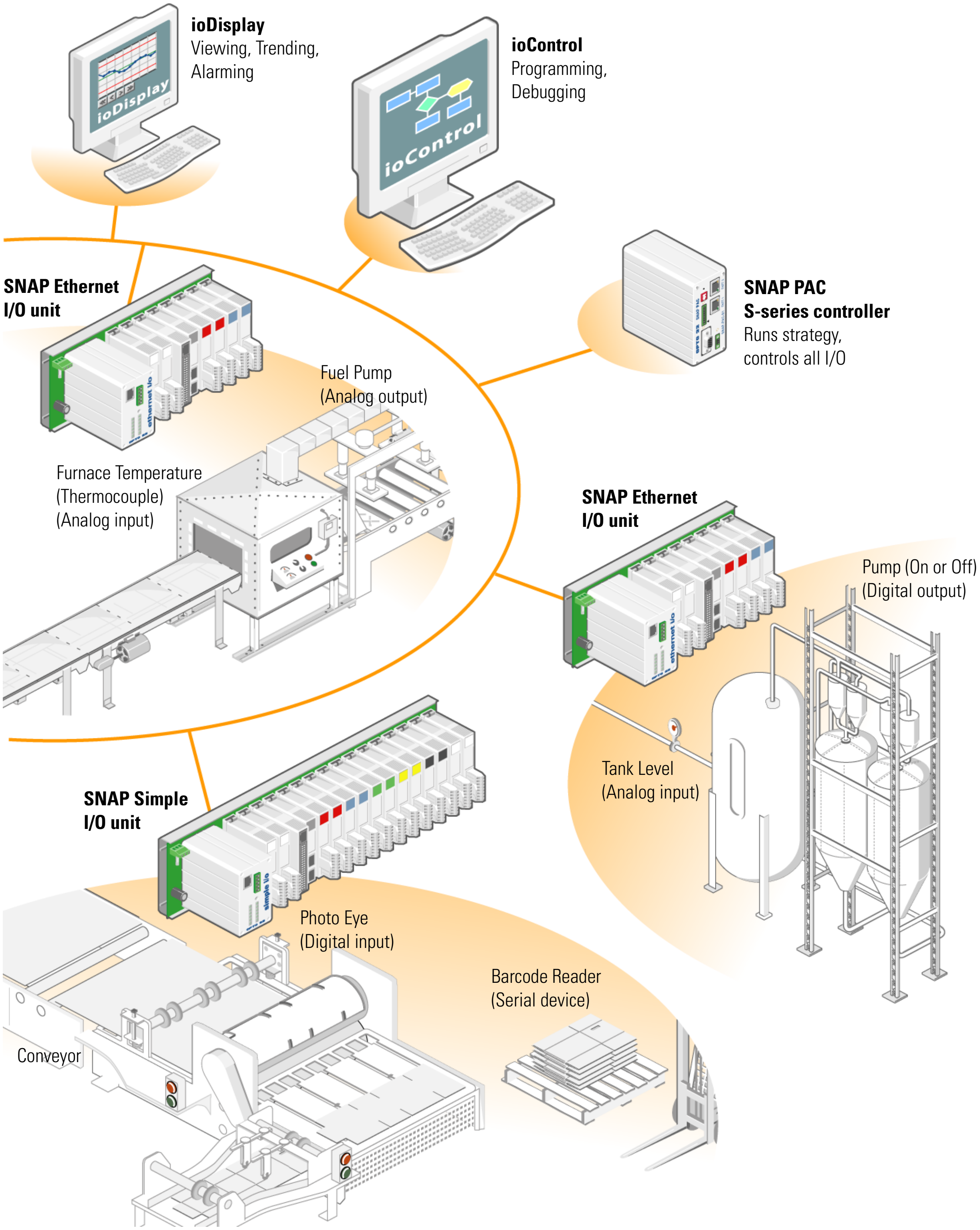
In this Chapter

About ioControl

ioControl is a programming language based on flowcharts. Whether you have ioControl Basic or ioControl Professional, you use ioControl to develop software that monitors and controls all kinds of equipment and sensors, from a simple heating system to a complex factory. The software you develop controls the Opto 22 hardware that runs your heating system or factory.

The diagram below shows how ioControl on your PC can work with the hardware in your control system. This example shows a small system; yours may be even smaller or considerably larger and more complex. The diagram uses a SNAP-PAC-S1 as the control engine. Input/output (I/O) points on the subordinate SNAP Ethernet and SNAP Simple I/O units monitor and control the analog, digital, and serial devices connected to them. All these terms are defined in the following pages.

Control System Example



General Control Concepts

Automation

Automation is a way of adding intelligence to an industrial process. When you automate a process, you are less dependent on human action to carry out the process. The process generally becomes faster, more accurate, and more reliable. For example, take a look at the tank level and pump combination in the diagram on the previous page. Instead of having a person watch the level in the tank and turn the pump on or off when the level gets too low or too high, you can automate the process by installing a processor and I/O. The processor and I/O respond in milliseconds and are on the job 24 hours a day.

Control Engines

These processors are programmable electronic components that provide the intelligence required for automation. In an Opto 22 ioControl system, the processor is called a *control engine*, and it is built into the Opto 22 controller.

Using ioControl, you create a set of instructions (a software program) that tells the control engine how every aspect of a process should work. Using ioControl, you download the software program to a Opto 22 SNAP PAC controller, and the controller runs it as a stand-alone application. Since the instruction set is stored in the processor's electronic memory, much as a small computer would store it, the PC can be turned off or used for other operations while the control engine in the processor runs the program. And the instructions can be easily modified when necessary.

In the diagram on the previous page, one SNAP PAC S-series controller runs the program that controls the three areas of automation.

Digital and Analog Inputs and Outputs

An industrial process can include many different hardware components: switches, pumps, tanks, valves, furnaces, conveyors, photo eyes, thermocouples, and so on. All the components communicate with the control engine in the controller by way of input/output (I/O) points.

Input points are wired to hardware that brings information into the control engine from the process. Examples of devices that can be wired to input points are thermocouples, switches, and sensors. The control engine takes the information from the input points—such as whether a switch is on or what temperature is registered on a sensor—processes it using the software instruction set, and returns information to the process through output points.

Output points are wired to hardware that receives information from the control engine and uses this information to control components of the process. For example, lights, motors, and valves are all devices that can be wired to output points. Using an output point, the control engine might turn on a light or open a valve.

There are two types of I/O points, digital and analog:

- **Digital points** can be either on or off (True or False). Push buttons and LEDs are examples of digital devices. An LED is either on or off; it has no other possible state.

  In the diagram on page 3-2, the photo eye is an example of a digital input device. The photo eye is either on or off. When it turns off as the sheet passes through its beam, the digital I/O module tells the control engine it is off, and the control engine responds as programmed to stamp the sheet.

  The pump is an example of a digital output device. Based on information from the tank level input, the control engine turns the pump on or off as programmed.

- **Analog points** have a range of possible values. Temperature and pressure are examples of analog information. Temperature might be any number in a range, -2 or 31.65 or 70.1 or many other possible numbers.

  In the diagram on page 3-2, the tank level sensor is an analog input device. It registers the changing level of water in the tank and reports the level to the control engine, which responds by turning the pump on or off.

  The fuel pump is an example of an analog output device. Based on information about the temperature in the furnace, the control engine adjusts the pressure of fuel through the pump as programmed.

SNAP Special-Purpose I/O Modules

In the ioControl system, some I/O modules do not contain standard analog or digital input/Output Points, but are used for special purposes.

SNAP serial communication modules provide two channels of serial data. Through these channels data can be sent to and received from a serial device. In the diagram on page 3-2, the barcode reader is an example of a serial device.

SNAP high-density digital (HDD) modules provide 32 channels of digital input or 32 channels of digital output in one module. Any of the digital input or output devices shown in the diagram could be wired to HDD modules rather than standard 4-channel digital modules.

Key Features

See the following topics for some of ioControl key features:

- "Copying I/O Configurations" on page 6-3

  A configuration file allows you to copy an I/O configuration from one strategy to another. Or, using ioManager you can send the configuration to multiple I/O units at the same time.

- "Using Network Segmenting in ioControl" on page 5-5

  Using the two independent Ethernet network interfaces on a SNAP PAC controller, you can segment the control network from the company network.

- "Using Ethernet Link Redundancy in ioControl" on page 5-6

  (ioControl Pro only) If you are using SNAP PAC controllers and ioControl Professional, you can use Ethernet link redundancy to set up alternate network links.

- "Configuring Event/Reactions" on page 6-42

  (ioControl Pro only) For *serial-based mistic I/O units* you can configure specific reactions to events when they occur.

- "Persistent Data" on page 9-4

  You can configure the data in most variables to be persistent. The variable's value is saved in the controller's memory; it does not change when the strategy is run, stopped, or started, and it does not change if the strategy is changed and downloaded again.

- "Pointer Commands" on page 10-56

  For advanced programming, you can create **pointers** that store the memory address of a variable or some other ioControl item, such as a chart, an I/O point, or a PID loop. You can perform any operation on the pointer that you could perform on the object the pointer points to.

- "Setting Initial Values in Tables During Strategy Download" on page 9-10

  When you are adding table variables in ioControl, you can set all table elements to one initial value. Or, you set each individual table element to its own value by creating an initialization file to download with your ioControl strategy.

- "Using Subroutines" on page 12-1

  You can use subroutines that are independent from strategies, but that can be called from any strategy. Subroutines offer two ways to work with variables and other logical elements: they can be passed in or they can be local to the subroutine.

- "Configuring PID Loops" on page 6-29

  A proportional integral derivative (PID) control system (often referred to as a PID loop) monitors a process variable, compares the variable's current value to a desired value (a setpoint), and calculates an output to correct error between the setpoint and the variable. Because the calculation is complex, it is done by a mathematical formula, or algorithm, that you can then adjust (tune) for each PID loop.

ioControl Terminology

Analog Point

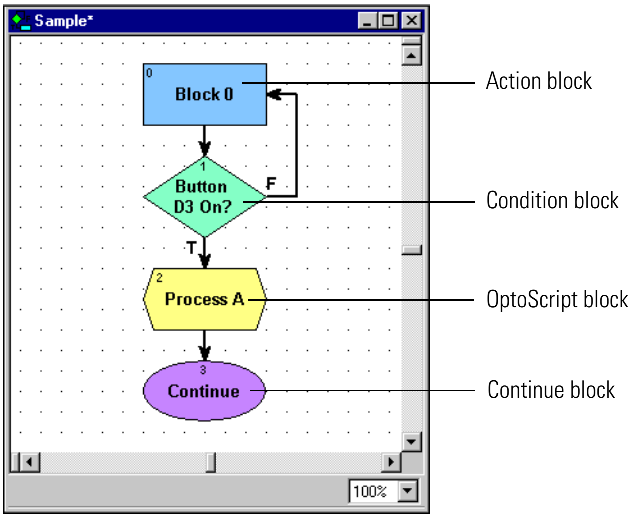
Analog points have a range of possible values. Temperature and pressure are examples of analog information. Temperature might be any number in a range, -2 or 31.65 or 70.1 or many other possible numbers. Analog points can be either inputs (such as a tank level sensor) or outputs (such as a fuel pump).

Blocks

A chart is made up of action, condition, continue, and OptoScript blocks connected by arrows, which show how the process flows.

Action Block

Rectangular blocks in a flowchart that contain one or more instructions (actions) that do the work of the strategy, such as turning things on or off, setting variables, and so on. Several instructions can be placed in one action block. Action blocks can have many entrances, but only one exit.



Condition Block

Diamond-shaped blocks in a flowchart that contain questions (conditions) that control the logical flow of a strategy. Condition blocks can have many entrances, but only two exits: True and False. The block can contain more than one condition, and you can use AND or OR to indicate whether all conditions must be true to exit True, or whether any one condition must be true to exit True.

Continue Block

Oval-shaped blocks in a flowchart that route the flow of execution to an action, condition, or OptoScript block. They do not contain any instructions, but store only the name of the next block to execute. Continue blocks can have many entrances, but no exits, since the exit is defined within the block. Continue blocks avoid awkward connections between two blocks that are far apart.

OptoScript Block

Hexagonal blocks in a flowchart that contain OptoScript code. OptoScript is a procedural language that can simplify tasks such as math computations, string handling, and complex loops and conditions. OptoScript blocks can have more than one entrance but only one exit.

Digital Point

Digital points can be either on or off (true or false). Push buttons and LEDs are examples of digital devices: an LED is either on or off; it has no other possible state. Digital points can be either inputs (such as a photo eye) or outputs (such as a pump).

External Value

The external value (XVAL) is the real-world value measured or set by an I/O point. The ioControl strategy reads and writes only to internal values, and transfers internal values to external values if the associated I/O unit is enabled.

Flowcharts

Since most control applications are complex, the strategy typically consists of a series of process flowcharts, or *charts*, that all work together. Each chart controls one aspect of the strategy—one piece of the automated process. Together, all the charts constitute the strategy. The total number of charts in a strategy is limited only by the amount of memory available in the control engine.

A chart can be running, suspended, or stopped. A *running chart* is actively performing its assigned task. A *suspended chart* is temporarily paused. A *stopped chart* is inactive. Every chart in an ioControl strategy can change the status of any other chart in the strategy, yet every chart is independent of every other chart. Any combination of charts can be running simultaneously, up to the maximum limit allowed on the control engine. (See also, "Multitasking" on page 3-8.)

Every strategy automatically contains a Powerup chart. The Powerup chart is automatically started when the strategy begins running, so it starts other charts. All other charts you create, based on the needs of your process.

Input Point

Input points are wired to hardware that brings information into the brain from the process. Examples of devices that can be wired to input points are thermocouples, switches, and sensors. The control engine in the brain takes the information from the input points—such as whether a switch is on or what temperature is registered—processes it using commands in the strategy, and returns information to the process through output points.

- A **floating point** (or float) is a numeric value that contains a decimal point, such as 3.14159, 1.0, or 1234.2. A good example of a float variable is one that stores readings from an analog input, such as a thermocouple.

- An **integer** is a whole number with no fractional part. Examples of integer values are -1, 0, 1, 999, or -456. The state of a switch, for example, could be stored in an integer variable as 1 (on) or 0 (off).
- A **timer** stores elapsed time in units of seconds with resolution in milliseconds. Up timers count up from zero, and down timers start from a value you set and count down to zero. For example, you could set a down timer to make sure a value is updated at precise intervals.
- A **string** stores text and any combination of ASCII characters, including control codes and extended characters. For instance, a string variable might be used to send information to a display for an operator to see.

  A string variable can contain numeric characters, but they no longer act as numbers. To use them in calculations, you must convert them into floating point or integer numbers. And a numeric value to be displayed on a screen must be converted into a string first.
- A **pointer** does not store the value of a variable; instead, it stores the memory address of a variable or some other ioControl item, such as a chart or an I/O point.
- **Communication handles** store parameters needed for communication with other devices.

You can use variables that are individual pieces of information, and you can also use *table variables*, which are groups of related information in the form of a table.

Instructions (Commands)

ioControl commands, or instructions, tell the control engine what to do at each step in the flowchart to control the process. Each block in a chart contains one or more instructions, such as *Convert Number to String* or *Start Counter* or *Chart Running?*

Commands are in two forms: Actions and Conditions.

- **Action commands** do something in the process; for example, *Convert Number to String* and *Start Counter* are both action commands. On the flowchart they appear as instructions in Action Blocks, which are rectangular in shape. They may also appear in hexagonal OptoScript Blocks.
- **Condition commands** check a condition and are in the form of a question. *Chart Running?* and *Variable False?* are examples of condition commands. They appear as instructions in Condition Blocks, which are diamond-shaped, or in hexagonal OptoScript blocks. Condition commands are questions that always have two possible answers, either yes or no (true or false). The answer determines the flow of logic and what happens next in the chart.

Internal Value

The internal value (IVAL) is the value read or written by the ioControl strategy. Internal values are transferred to external values only when the I/O unit is enabled.

Multitasking

The control engine can run several charts seemingly at once, each performing a different task, through a time-slicing technique called multitasking (also called multicharting). Opto 22 SNAP Ultimate I/O control engines can run up to eight charts plus one host task simultaneously; SNAP PAC R-series and SNAP-LCE controllers can run up to 16 charts plus one host task simultaneously; SNAP-PAC S-series controllers can run up to 32 charts plus one host task simultaneously. The host task is an invisible chart used to communicate to a PC, which may be running ioControl in Debug mode or ioDisplay.

Each chart in a running or suspended state counts toward the total that can run simultaneously. Charts that are stopped do not. When the Powerup chart is running, it also counts.

The actual order and timing for running tasks is not deterministic—that is, it is not always the same, but depends on priorities at any given time. For example, communication may sometimes take a higher priority than a running chart.

Output Point

Output points are wired to hardware that receives information from the brain and uses this information to control components of the process. For example, lights, motors, and valves are all devices that can be wired to output points. Using an output point, the control engine in the brain might turn on a light or open a valve.

Pointer

A pointer does not store the value of a variable; instead, it stores the memory address of a variable or some other ioControl item, such as a chart, an I/O point, or a PID loop. You can perform any operation on the pointer that you could perform on the object the pointer points to. Pointers are an advanced programming feature and are very powerful, but they also complicate programming and debugging.

Because pointers can point to any data type, pointer tables can store an assortment of data types in a single table.

Pointers allow one level of indirection within ioControl; a pointer cannot point to another pointer. After you add a pointer through the Add/Edit Variable Dialog Box, you can use it wherever the object itself would be used.

Strategy

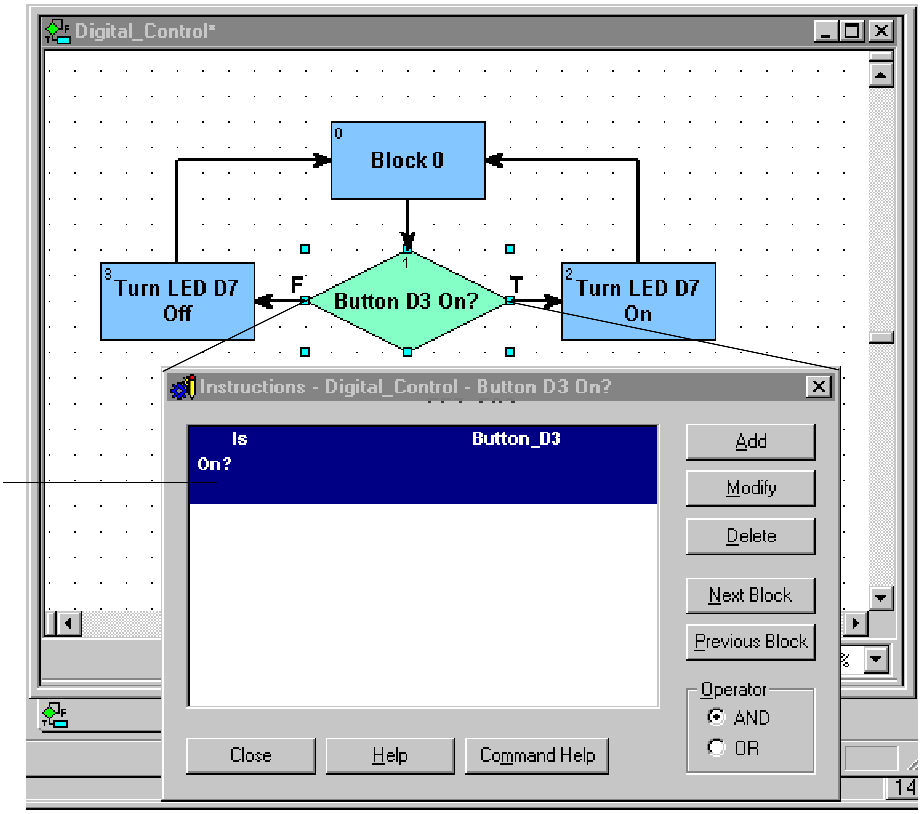
The software program you create using ioControl Basic or ioControl Professional. The strategy includes all the definitions and instructions (commands) necessary to control the process that one Opto 22 controller handles. Since most control processes are complex, the strategy typically consists of a series of process flowcharts that all work together, with each chart controlling one piece of the automated process. You may have several ioControl systems, each controlling a different process and therefore running different strategies. Or you may have two or more ioControl systems controlling identical processes in different areas and running the same strategy.

Variables

A variable is a holding place that represents a piece of information in a strategy, such as the parameters for communication, temperature reported by a thermocouple, the name of a chart, or a group of words and numbers to be sent to a display. The information a variable represents is called the *value* of the variable. As a strategy runs, the variable's name remains the same, but its value may change. For example, the value of a variable named Oven_Temperature may change several times while its strategy is running, but its name remains Oven_Temperature.

A variable stores one of six types of data: floating point, integer, timer, string, pointer, or communication handle. When you create the variable, you designate the type of data it contains.

The instruction shown below is for the condition block, *Button D3 On?* This block contains only one instruction. As you look at the chart, you can see that the answer to the question in the instruction determines whether the process flows to *Turn LED D7 On* or to *Turn LED D7 Off.*

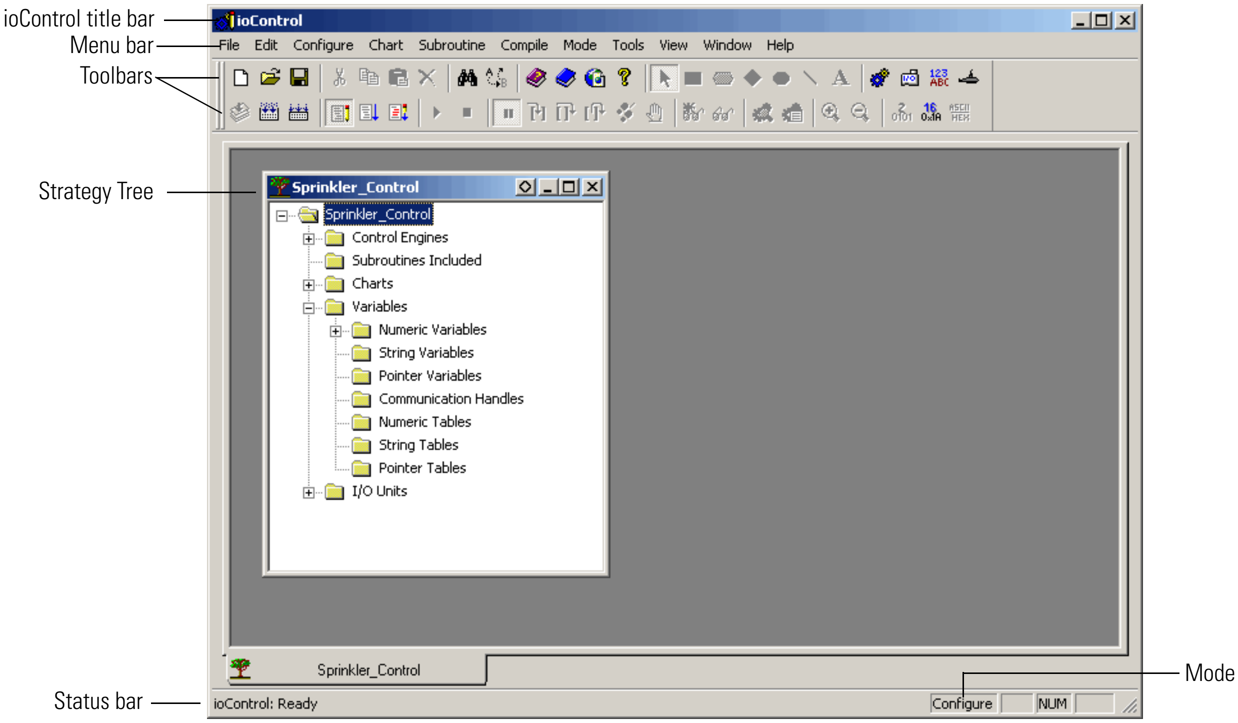


Is Button_D3 On?

Instruction for the block named Button D3 On?

ioControl Main Window

With a strategy open, the main window in ioControl Basic and ioControl Pro looks similar to this:



Since ioControl uses standard Microsoft Windows conventions, you'll recognize the title bar and menu bar and already be familiar with some of the menus, such as File, Edit, View, Window, and Help. This section discusses some things that may not be familiar.

Status Bar

The status bar at the bottom of the window shows you information about ioControl. When you move your mouse over the toolbar, a description of the button you're pointing to appears in the status bar. The status bar indicates the mode ioControl is in and indicates messages or errors helpful in debugging.

To hide or show the status bar, choose View→Status Bar. A check mark next to the menu item means the status bar will show; no check mark means it is hidden.

Mode

You can run ioControl in three modes: Configure, Debug, or Online. The current mode is shown in the status bar, on the right. Toolbars and menus change depending on the mode.

- **Configure mode** is used to create, modify, save, and compile strategies, flowcharts, and subroutines; and to configure control engines, I/O, and variables.

- **Debug mode** is used to download, run, and debug strategies and to view control engine and communication status and errors while the Strategy is running.
- **Online mode** is a scaled-down version of Configure mode, used to change a strategy while it is running. You cannot add variables and I/O in Online mode, but you can change ones that already exist.

  NOTE: When you change a chart in Online mode, a new copy of that chart is downloaded to the control engine, but the old one is not deleted. After you have made a few online changes, the additional chart copies begin to take up memory. To avoid memory problems, be sure you stop the strategy after making several online changes and completely compile and download it to clear out old chart copies.

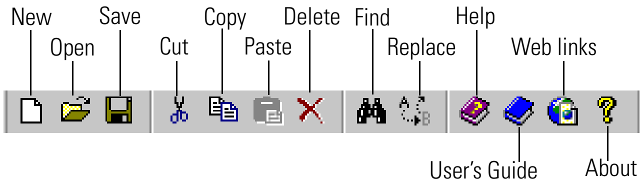
To change modes, choose the mode you want from the Mode menu, or click its button in the toolbar.

Toolbars

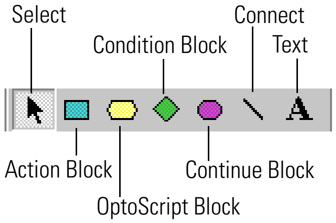
Toolbars give you shortcuts for doing many things that also appear on menus. Toolbars in ioControl include standard Windows buttons like New and Help as well as special buttons for ioControl functions. Like the menu bar, the tools that you can use depend on which mode you're in. Tools that don't apply to a mode are grayed out.

The following toolbars are standard in ioControl. To change the buttons in them or to create your own toolbar, see "Customizing Toolbars" on page 3-25.

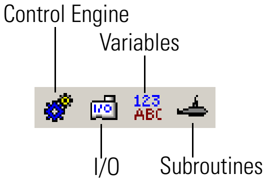
File Toolbar



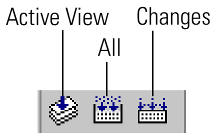
Drawing Toolbar



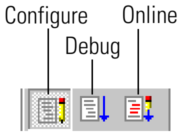
Configure Toolbar



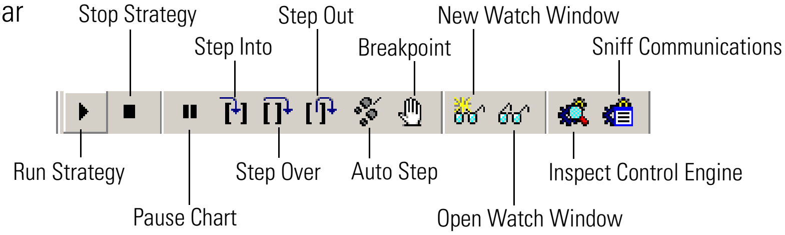
Compile Toolbar



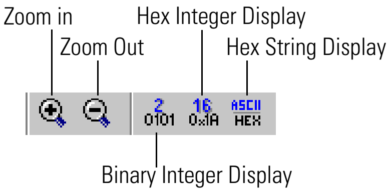
Mode Toolbar



Debug Toolbar



View Toolbar

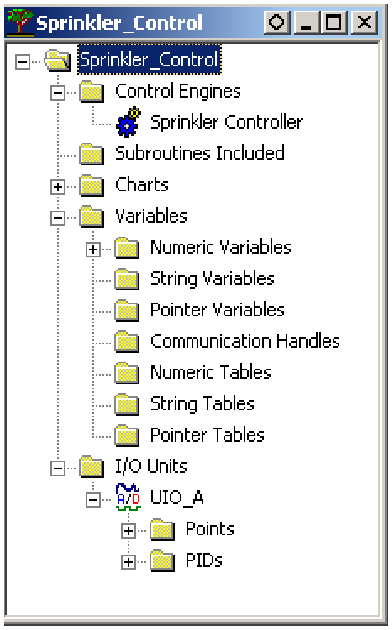


Moving Toolbars

You can move toolbars to the place where they are most convenient for you. To move a toolbar, click and drag it with the mouse to the location where you want it to be.

Hiding and Showing Toolbars

To hide or show a toolbar, choose View→Toolbars. In the dialog box, click to put a check mark in the box next to the toolbar you want to show. To hide a toolbar, click the box to remove the check mark.



- Sprinkler_Control
  - Control Engines
    - Sprinkler Controller
  - Subroutines Included
  - Charts
  - Variables
    - Numeric Variables
    - String Variables
    - Pointer Variables
    - Communication Handles
    - Numeric Tables
    - String Tables
    - Pointer Tables
  - I/O Units
    - UIO_A
      - Points
      - PIDs

Strategy Tree

The Strategy Tree (shown at left) opens when you open a Strategy, and closing it is equivalent to closing the strategy. The Strategy Tree shows you all the elements of your strategy: control engines, flowcharts, subroutines, variables, I/O units and points, and PIDs.

The Strategy Tree works just like Windows Explorer: you can expand or collapse folders to view or hide what is in them. You can easily see what is in your strategy, open elements to change them by double-clicking them, or open a pop-up menu by right-clicking on an element.

Each element in the strategy is represented by a button, shown just to the left of its name. The table below shows the buttons and what they represent.

Button	Description	Button	Description
	Control Engine	123 32	Integer 32 Table
	Chart	123 64	Integer 64 Table
	Subroutine	ABC	String Table
840 32	Integer 32 Variable		Pointer Table
64 257	Integer 64 Variable	D	Digital I/O Unit
3.14	Float Variable	A/D	Mixed I/O Unit
	Down Timer Variable	I←	Analog Input Point
	Up Timer Variable	I←	Digital Input Point
	String Variable	O→	Analog Output Point
⇄	Communication Handle	O→	Digital Output Point
3.14	Float Table	PID	PID Loop
☞	Pointer Variable		

Windows and Dialog Boxes in ioControl

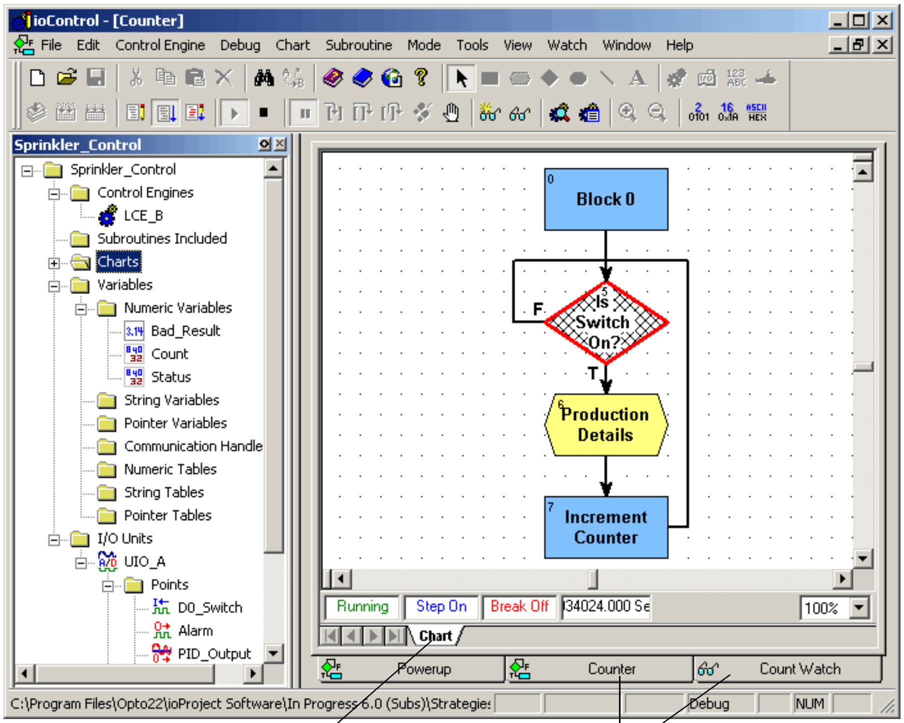
Windows and dialog boxes in ioControl follow Microsoft Windows standards. You can minimize, maximize, move, resize, and tile them as needed. See your Microsoft Windows documentation for information on how to do these things.

The following topics describe other useful features in ioControl:

- "Using Tabs to View Open Windows" on page 3-15
- "Docking Windows" on page 3-17
- "Splitting a Chart or Subroutine Window" on page 3-18
- "Zooming in a Chart or Subroutine Window" on page 3-20
- "Redrawing a Chart or Subroutine Window" on page 3-21
- "Changing Column Width in a Dialog Box" on page 3-22
- "Sorting Data in a Dialog Box" on page 3-23

Using Tabs to View Open Windows

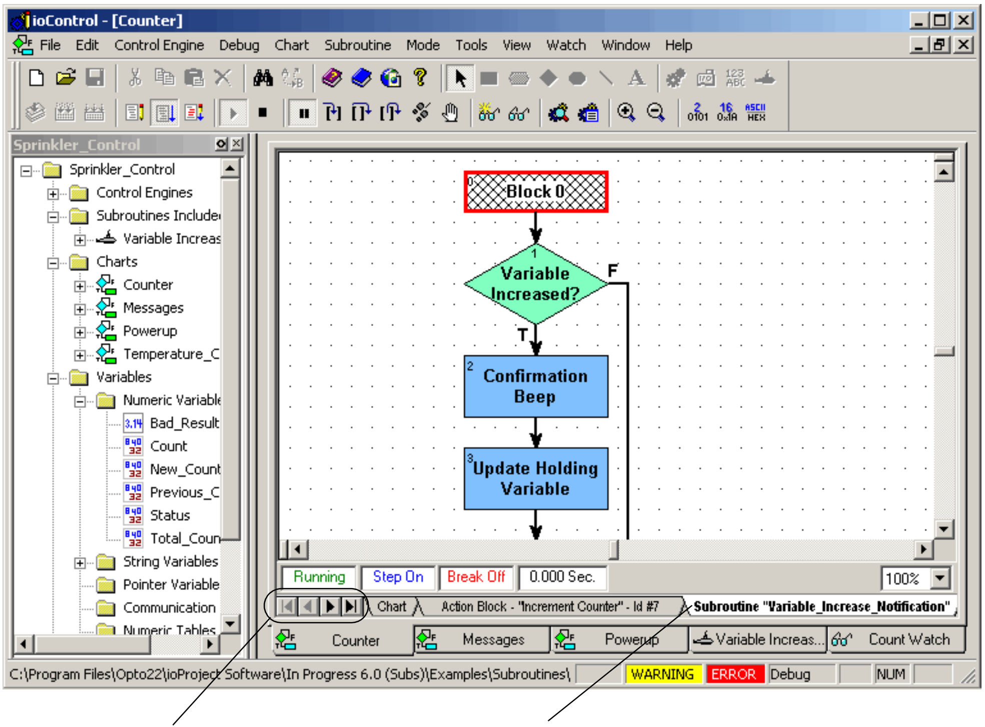
When multiple windows are open—especially if they are maximized—it can be difficult to know where you are, and windows can become lost behind each other. However, you can click the tabs at the bottom of the main ioControl window to move among chart windows, subroutine windows, watch windows, and blocks you may be stepping through for debugging.



White tabs show where you are when stepping through a chart or subroutine.

Gray tabs show open windows. Click a tab to bring its window into view.

The upper layer of tabs appears when you are stepping through your Strategy in Debug mode. It acts as a kind of call stack to let you see how you got to the current block or command.



Click the arrow buttons to see tabs that are not visible or are only partly visible.

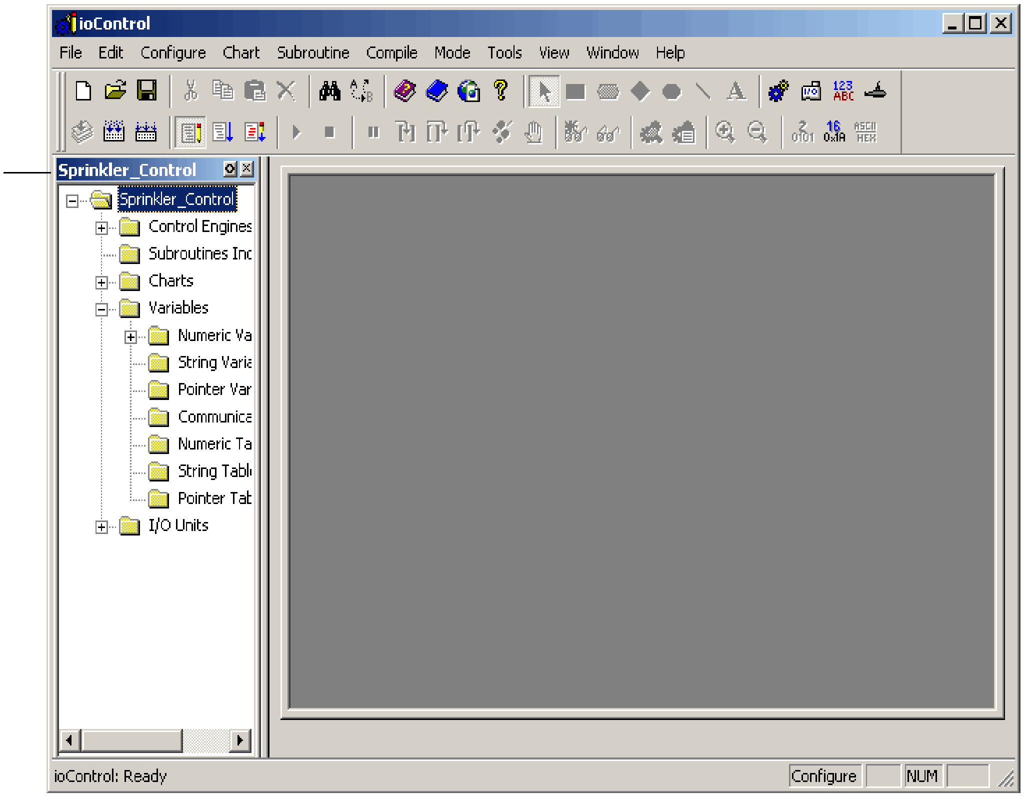
This white tab shows you are stepping inside the *Variable_Increase_ Notification* subroutine, which was called by the *Increment Counter* action block on the gray tab at left. The gray *Chart* tab farther left shows the chart this action block is in. Click any tab to see how you got to where you are.

Docking Windows

You can place the Strategy Tree and watch windows where you want them (“dock” them) in the ioControl main window. Docked windows do not have tabs but are contained in their own frames. ioControl remembers their position the next time you open that strategy.

- **To dock a window**, click the docking button ⊙ in the window’s title bar.

The window moves to its own frame. (Compare the following figure with the one on page 3-11 to see the difference in the Strategy Tree window.)

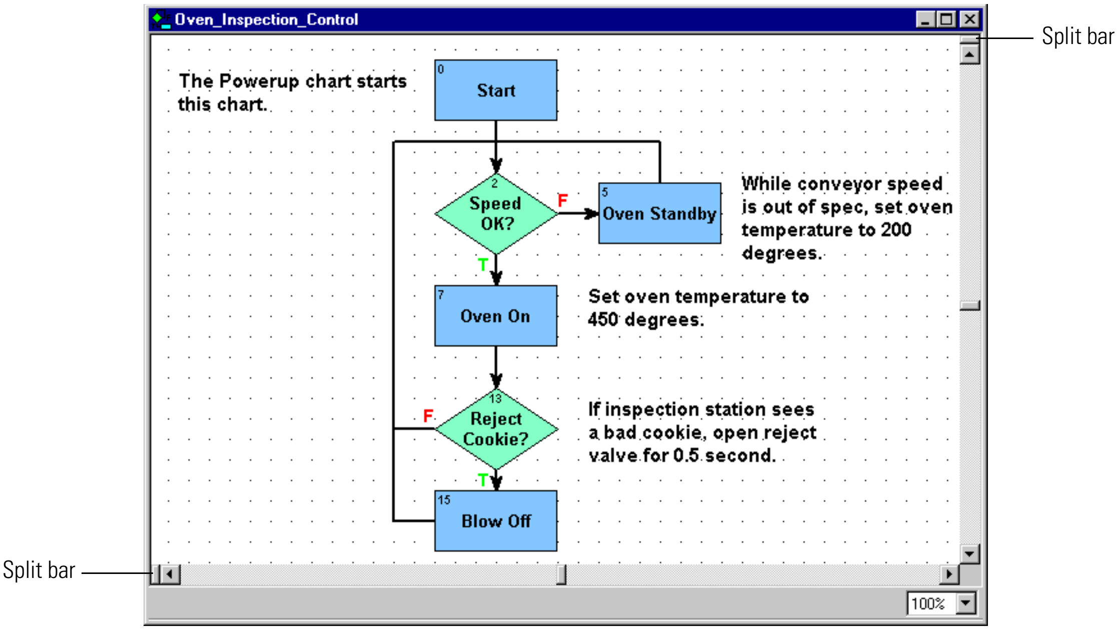


- **To change the docked position**, click and drag the window's title bar to any edge of the main window.
- **To change the docked window's width or height**, click and drag the edge of its frame in the direction you want.
- **To free the window** from its docked position, click the docking button ⚙ in its title bar.

Splitting a Chart or Subroutine Window

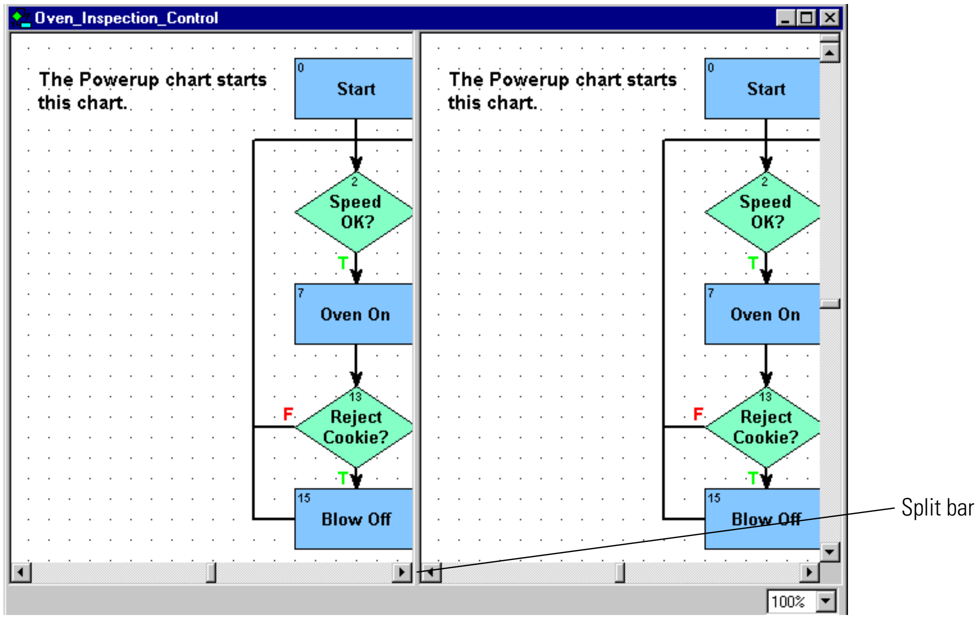
In chart and subroutine windows, you can split the view of a window to see two horizontal views, two vertical views, or even four views of the same chart or subroutine. You can scroll around within each view to compare parts or to copy and paste elements. Splitting is especially useful for large charts and subroutines.

Split bars appear at the top of the right scroll bar and at the left of the bottom scroll bar.


Oven_Inspection_Control
Split bar
The Powerup chart starts this chart.
0 Start
2 Speed OK?
F
5 Oven Standby
While conveyor speed is out of spec, set oven temperature to 200 degrees.
T
7 Oven On
Set oven temperature to 450 degrees.
13 Reject Cookie?
F
If inspection station sees a bad cookie, open reject valve for 0.5 second.
T
15 Blow Off
Split bar
100%

- **To divide a window vertically**, double-click the split bar at the left of the bottom scroll bar, or click and drag the split bar to the right.

  The window is split into two vertical views of the same chart or subroutine.



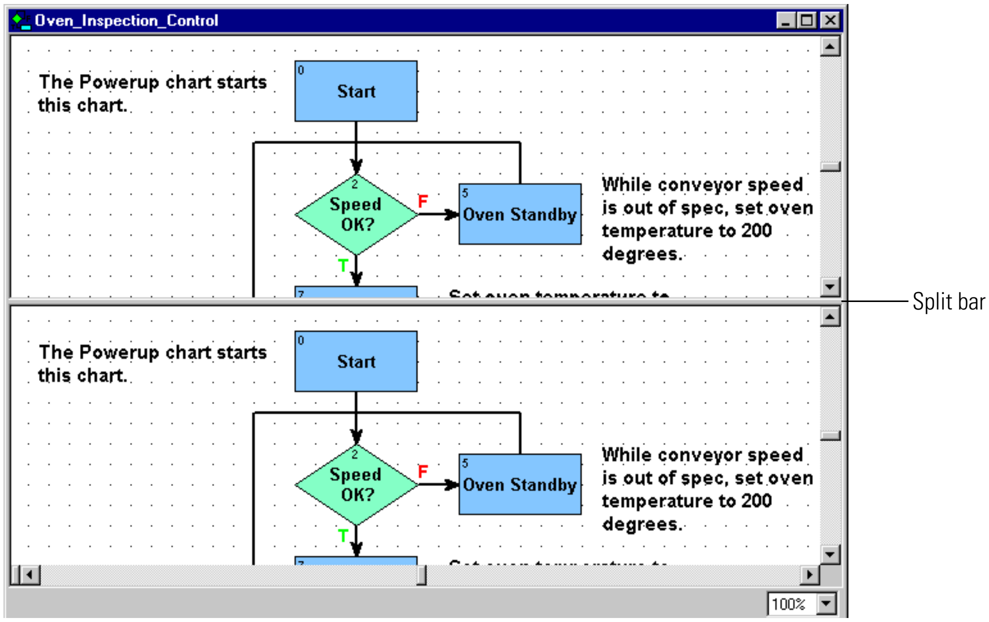
- **To scroll in any view**, click in the view to make it active, then use the scroll bars as usual.

  You can also zoom in the active view. (See page 3-20 for more information on zooming.)

- **To return the window to full view,** double-click the split bar again or click and drag it back to the left of the scroll bar.

- **To divide a window horizontally**, double-click the split bar at the top of the right scroll bar, or click and drag the split bar down.

  The window is split into two horizontal views of the same chart or subroutine.


Oven_Inspection_Control
The Powerup chart starts this chart.
Start
Speed OK?
F
T
Oven Standby
While conveyor speed is out of spec, set oven temperature to 200 degrees.
Split bar
100%

- **To return the window to full view,** double-click the split bar again or click and drag it back to the top of the scroll bar.

You can divide a window into four views by splitting horizontally and vertically at the same time.

Remember that when you split a window, you are looking at multiple views of *the same* chart or subroutine. You do not create two different charts/subroutines when you split a window.

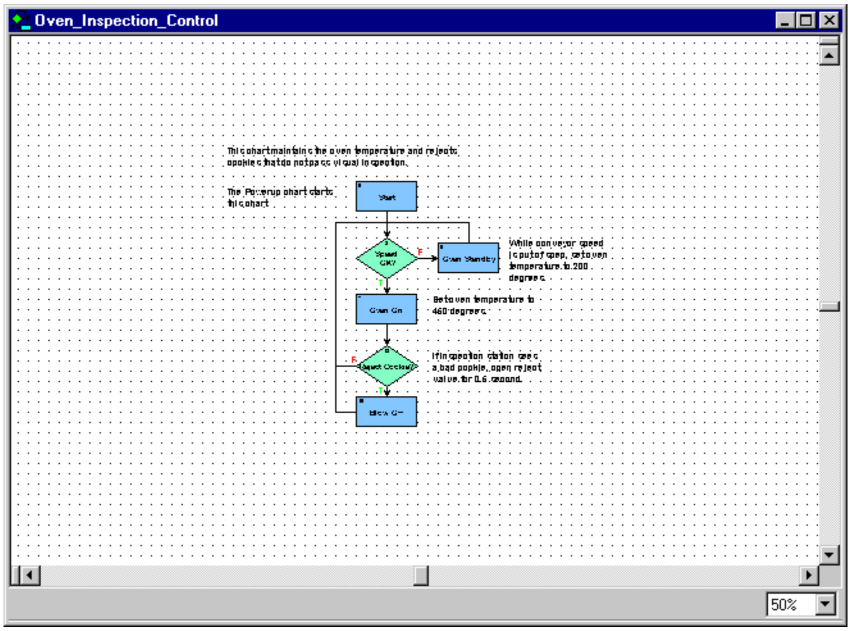
Zooming in a Chart or Subroutine Window

You can change the scale of any chart or subroutine window by using the zoom feature. You can view elements at seven sizes, from one-eighth to eight times the normal size. There are several ways to zoom:

- Click the Zoom In or Zoom Out buttons on the toolbar.
- Press the + or - keys on the keyboard.
- Right-click anywhere on a chart or subroutine window to display a pop-up menu. Select Zoom and choose In (to zoom in at twice the size), Out (to zoom out at half the size), or Reset (to return the zoom to 100 percent).
- From the Window menu, choose Zoom In, Zoom Out, or Zoom Reset.
- To pick the zoom percentage you want, click the Zoom field at the bottom right of the window and select 12.5 percent, 25 percent, 50 percent, 100 percent (the default), 200 percent, 400 percent, or 800 percent.

NOTE: Zooming always takes place with reference to the center point of the window.

Here is a window at 50 percent zoom:



This chart maintains the oven temperature and rejects cookies that do not pass visual inspection.

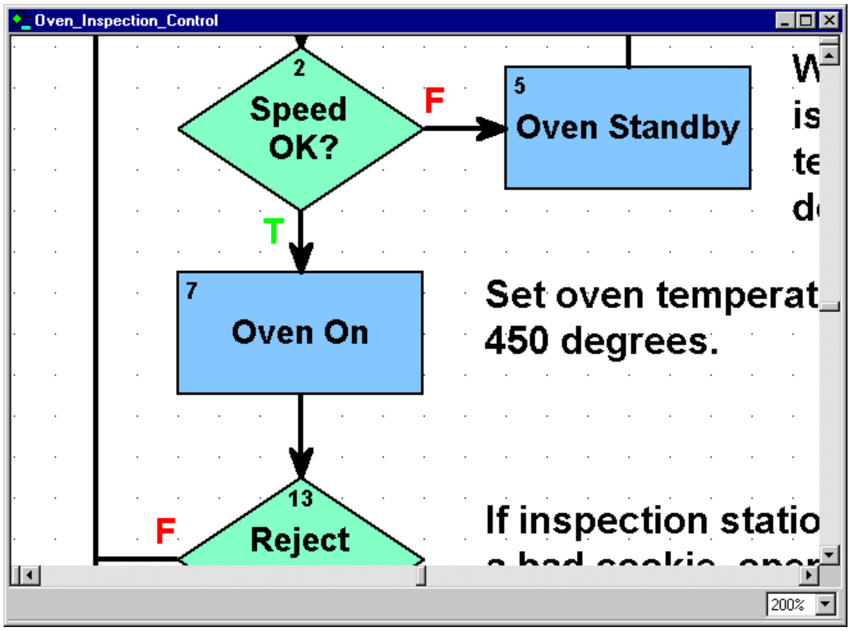
The Powerup chart starts this chart

While conveyor speed is out of spec, set oven temperature to 200 degrees.

Set oven temperature to 450 degrees.

If inspection station sees a bad cookie, open reject valve for 0.5 second.

The same window at 200 percent zoom looks like this:



Set oven temperat
450 degrees.

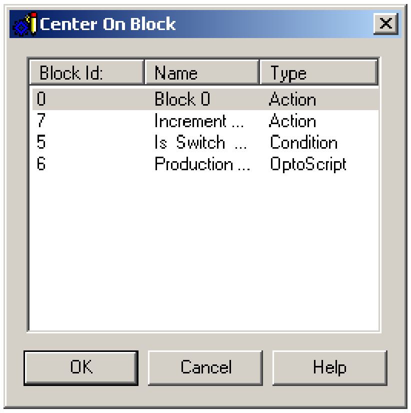
If inspection statio
a bad cookie, open

Redrawing a Chart or Subroutine Window

If you want to move quickly to a particular block, you can redraw a chart or subroutine window with any block at its center.

1. With a chart or subroutine open in the active window, select View➞Center On Block, or right-click in the chart or subroutine window and choose Center On Block from the popup menu.

   The Center On Block dialog box appears, listing all blocks in the chart.



Block Id:	Name	Type
0	Block 0	Action
7	Increment ...	Action
5	Is Switch ...	Condition
6	Production ...	OptoScript

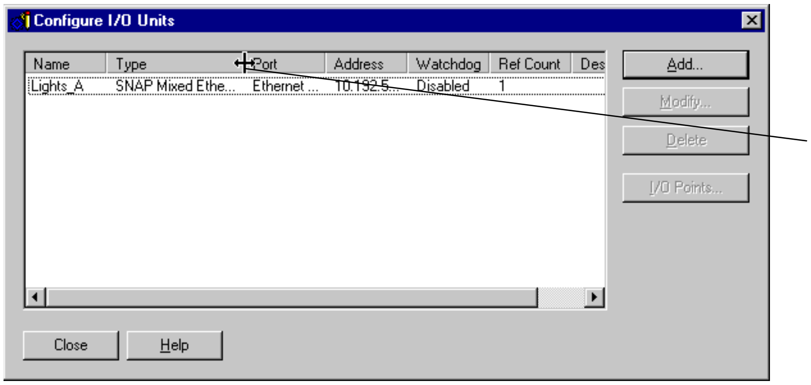
2. Double-click the block you want, or click it once and click OK.

   The chart or subroutine is redrawn with the selected block in the center of the window.

Changing Column Width in a Dialog Box

Many dialog boxes include several columns of information. To see all the data in some columns, you may need to make columns wider.

- **To widen or narrow a column**, click the right edge of the column label and drag it horizontally.



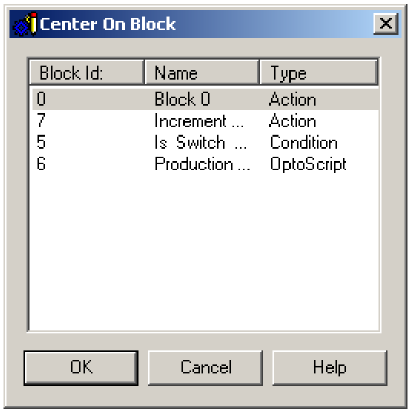
Name	Type	Port	Address	Watchdog	Ref Count	Des
Lights_A	SNAP Mixed Ethe...	Ethernet ...	10.192.5...	Disabled	1	

Click and drag

- **To resize a column to the exact width of its longest entry**, double-click the line that separates the column from the one on its right.

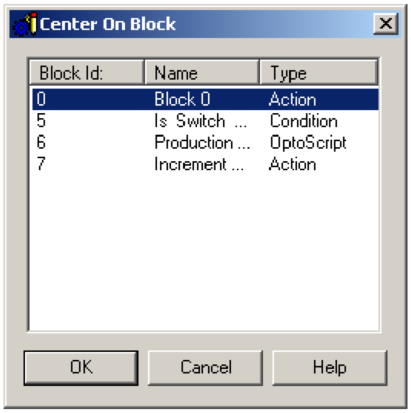
Sorting Data in a Dialog Box

In some dialog boxes with columns of data, you can sort the information in the way you want to see it. The Center On Block dialog box provides an example. The blocks in this dialog box normally appear in alphabetical order by the block name.



Block Id:	Name	Type
0	Block 0	Action
7	Increment ...	Action
5	Is Switch ...	Condition
6	Production ...	OptoScript

- **To sort by block number** instead, click the Block Id column label.



Block Id:	Name	Type
0	Block 0	Action
5	Is Switch ...	Condition
6	Production ...	OptoScript
7	Increment ...	Action

- **To sort in the opposite order** (descending numbers instead of ascending numbers), click the Block Id column label again.

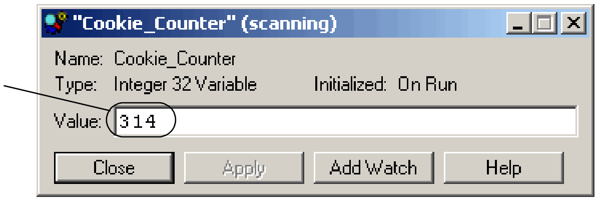
Some dialog boxes don't allow custom sorting. If you click a column label in these dialog boxes, nothing happens.

Customizing ioControl for Your Needs

Setting Decimal, Binary, or Hex Display Mode

You can set up ioControl to show the values of all integers and integer tables in decimal (the default), binary, or hexadecimal (hex) format. Binary and hex formats help you use masks. Hex format is particularly useful for entering integer literals and initial values for a digital-only Ethernet brain. Hex view is available in all modes: Configure, Debug, and Online.

The default view is in decimal notation, as shown in the following figure.

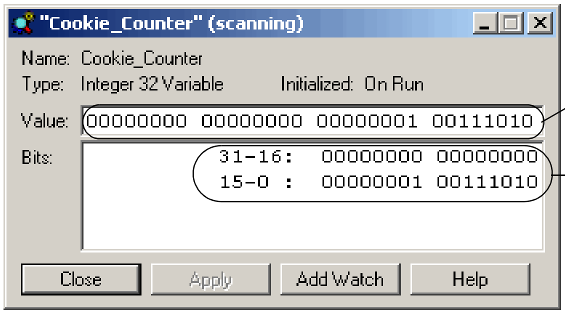


Decimal view

"Cookie_Counter" (scanning)

Name: Cookie_Counter
Type: Integer 32 Variable   Initialized: On Run
Value: 314

To view integers and integer tables in binary view, click the Binary Integer Display button on the toolbar or choose View→Binary Integer Display. The integers appear as shown below.



"Cookie_Counter" (scanning)

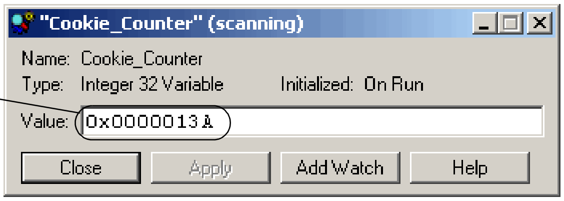
Name: Cookie_Counter
Type: Integer 32 Variable   Initialized: On Run
Value: 00000000 00000000 00000001 00111010
Bits:

```
31-16:   00000000 00000000
15-0 :   00000001 00111010
```

Binary view. All the bits cannot be shown at once.

Here you can see all the bits, shown in bytes so masks are easy to understand. Click on a line to see its place in the string.

- **To view integers and integer tables in hex view**, click the Hex Integer Display button on the toolbar or choose View→Hex Integer Display. Here's how the integers appear in hex:



Hex view. The 0x before the number indicates that the number is in hex.

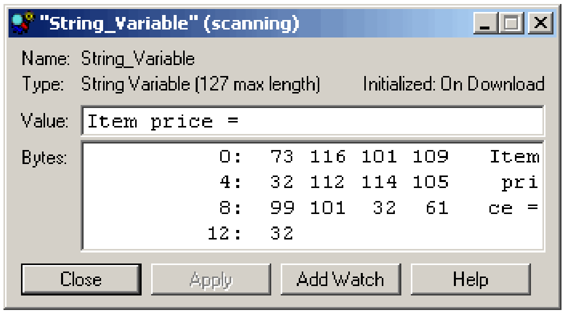
"Cookie_Counter" (scanning)

Name: Cookie_Counter
Type: Integer 32 Variable   Initialized: On Run
Value: 0x0000013A

- **To return to decimal view,** click the toolbar button again or choose the same item again from the View menu.

Setting Hex String View

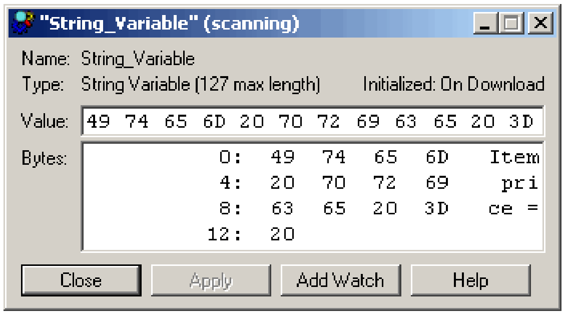
You can also set strings to appear in hex notation. Here is the View Variable window showing a string in regular notation:



Name: String_Variable
Type: String Variable (127 max length) Initialized: On Download
Value: `Item price =`

```
Bytes:   0:  73 116 101 109   Item
         4:  32 112 114 105    pri
         8:  99 101  32  61   ce =
        12:  32
```

To change to hex notation, click the Hex String Display button on the toolbar or choose View→Hex String Display. Here's how the string appears in hex:



Name: String_Variable
Type: String Variable (127 max length) Initialized: On Download
Value: `49 74 65 6D 20 70 72 69 63 65 20 3D`

```
Bytes:   0:  49  74  65  6D   Item
         4:  20  70  72  69    pri
         8:  63  65  20  3D   ce =
        12:  20
```

Customizing Toolbars

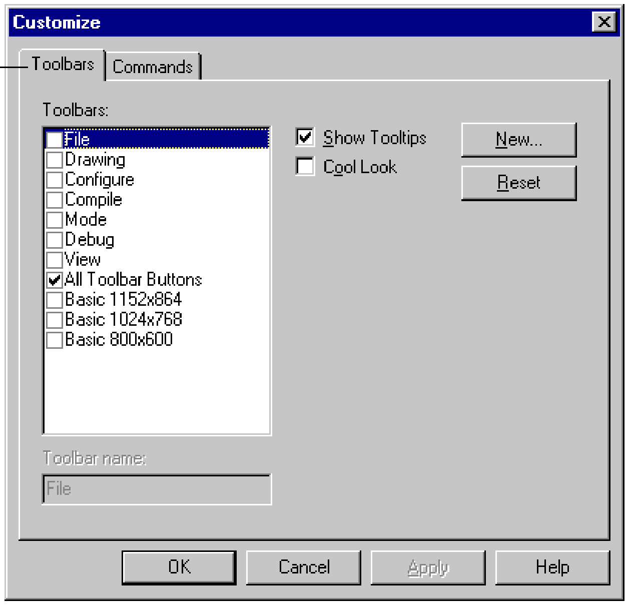
You can customize toolbars in ioControl for the way you work. You can choose toolbars to match your screen resolution. You can move toolbars to another place in the window. You can move a button into another position or toolbar, or delete it if you don't use it. You can even create your own toolbar with just the buttons you use.

Choosing Toolbars for Your Screen Resolution

You can choose toolbars to match your screen resolution and place the most frequently used toolbar buttons all on one line. To do so, follow these steps:

1. Choose View→Toolbars.

2. In the Customize dialog box, make sure the Toolbars tab is open.



Toolbars tab

Toolbars:

□ File
□ Drawing
□ Configure
□ Compile
□ Mode
□ Debug
□ View
☑ All Toolbar Buttons
□ Basic 1152x864
□ Basic 1024x768
□ Basic 800x600

☑ Show Tooltips
□ Cool Look

Toolbar name: File

3. Click to place a check mark next to the toolbar for your current screen resolution. Click OK.

   The most commonly used tools appear in a single line in the main window. If you want to change the tools, see “Moving and Deleting Buttons” on page 3-27.

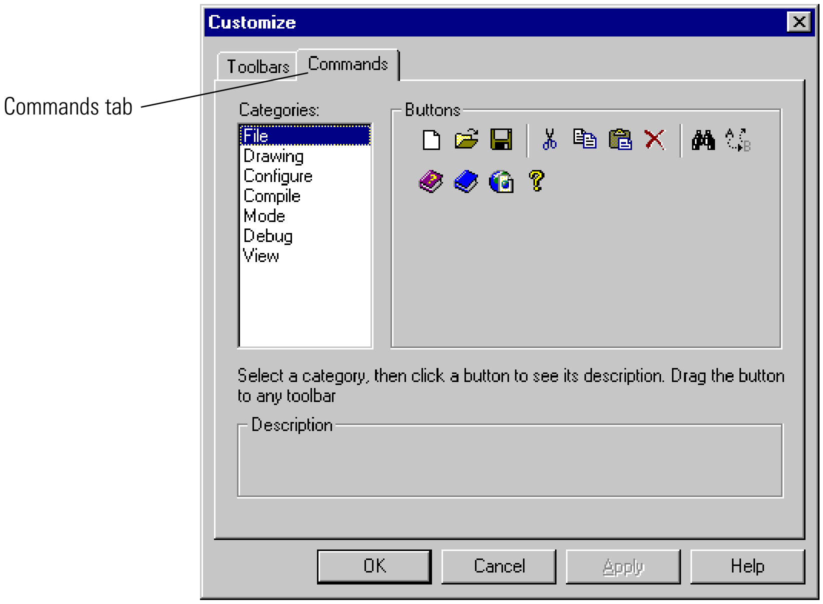
Moving Toolbars

To move a toolbar, click on its edge to outline the whole toolbar. Then drag the toolbar where you want it. You can place it at the top, bottom, left, or right of the main window or of any docked window.

Moving and Deleting Buttons

1. Choose View→Toolbars.

2. In the Customize dialog box, click the Commands tab.



Select a category, then click a button to see its description. Drag the button to any toolbar

3. In the Categories list, click the name of the toolbar you want to move the button from.

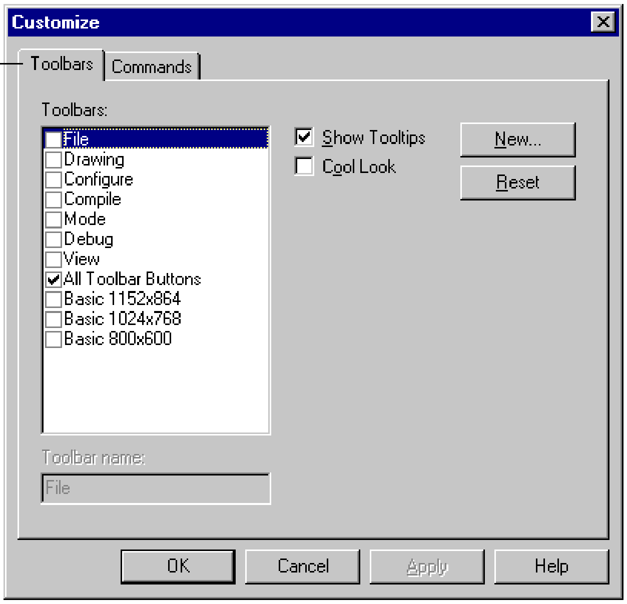
   The buttons in that toolbar appear in the Buttons area.

- **To change a button's position** in its current toolbar, click the button in the ioControl main window (not in the dialog box) and drag it to the position you want.
- **To move a button to another toolbar**, click the button either in the dialog box or in the main window, and drag it to the toolbar in the main window where you want it.
- **To delete a button from a toolbar**, click the button in the main window and drag it out of its toolbar.

Creating Your Own Toolbar

1. Choose View→Toolbars.

2. In the Customize dialog box, make sure the Toolbars tab is open.



Toolbars tab

Customize

Toolbars:

- □ File
- □ Drawing
- □ Configure
- □ Compile
- □ Mode
- □ Debug
- □ View
- ☑ All Toolbar Buttons
- □ Basic 1152x864
- □ Basic 1024x768
- □ Basic 800x600

☑ Show Tooltips

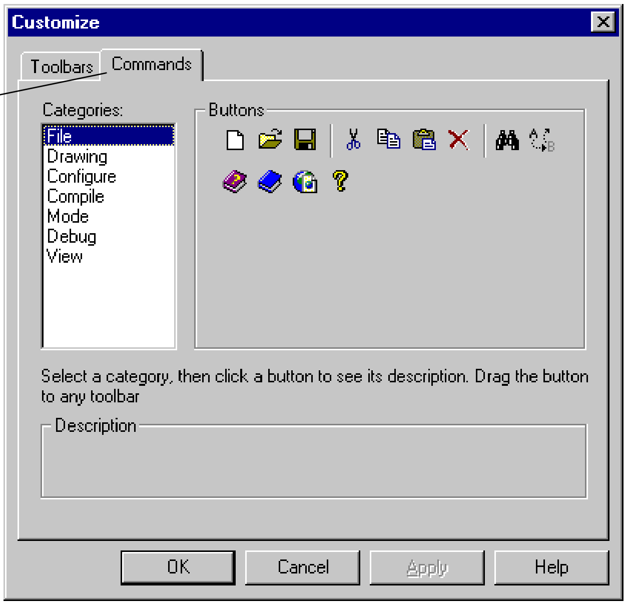
□ Cool Look

Toolbar name: File

3. Click New. Enter a name for the custom toolbar and click OK.

   A small box appears in the upper-left corner of the main window. You build your custom toolbar by placing the buttons you want in this box.

4. In the Customize dialog box, click the Commands tab.



Commands tab

Customize

Categories:

- File
- Drawing
- Configure
- Compile
- Mode
- Debug
- View

Select a category, then click a button to see its description. Drag the button to any toolbar

5. In the Categories list, click the name of the standard toolbar that contains the button you want to place in the custom toolbar.

   The buttons in that toolbar appear in the Buttons area.

6. Click the button you want and drag it to its place in the small gray box. Repeat until you have all the buttons you want in the custom toolbar.

7. When you have finished building the custom toolbar, click it to outline it, and drag it into position in the ioControl main window.

8. When you have finished customizing toolbars, click OK in the Customize dialog box.

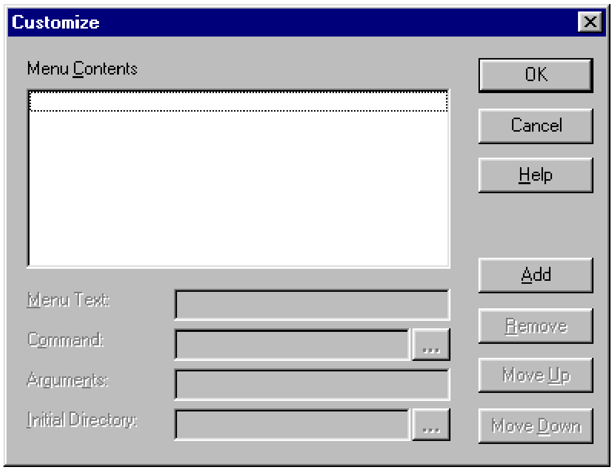
Setting Up Applications to Launch from ioControl

You may find it useful to launch other software applications directly from ioControl. For example, you may want to launch ioDisplay, Notepad, or the Calculator from ioControl. You can set up ioControl so that these applications appear in the Tools menu.

NOTE: If you launch an application from within ioControl when that application is already running, a second instance of the application may open. It depends on the application; some check to see whether the application is already running, and some do not.

1. With ioControl open, choose Tools→Customize.

   The Customize dialog box appears.



2. Click Add.

3. In the Menu Text field, type the name of the application as you want it to appear in the Tools menu.

4. In the Command field, type the path for the application's executable file, or click the browse button ... and navigate to the file.

5. (Optional) In the Arguments field, type any necessary command line parameters.
6. (Optional) In the Initial Directory field, type the directory the application should default to when it runs.

   For example, this is the directory the application would show when you open or save files.
7. Repeat the steps to add other applications. To change an application's position in the menu list, highlight it and click the Move Up or Move Down keys.
8. When you have finished adding applications, click OK.

   You return to the ioControl main window, and the applications you've added now appear in the Tools menu.

Online Help

To open online Help, choose Help➞Help Topics, or click the Help button in any dialog box. Help buttons in dialog boxes are context sensitive and provide help specifically on that dialog box. Buttons labeled "Command Help" give specific information on the command (instruction) you are currently using.

For brief explanations of buttons, move your mouse over the button and look in the status bar.

In Action and Condition Instructions dialog boxes, let your mouse rest for a second on an instruction (command), and you'll see a list of the variable types used in that command. (To show or hide the variable types list, in Configure mode, choose ioControl Options➞Show/Hide Instruction Type Information.)

To open online copies of ioControl manuals and quick reference cards, choose Help➞Manuals. You will need Adobe Acrobat Reader to open these files. You can also find information, documents, and support on the Opto 22 Web site by choosing Help➞Opto 22 on the Web.

CHAPTER 4

Designing Your Strategy

Introduction

This chapter introduces you to ioControl programming—how to design an ioControl strategy to control your automated process. For additional important information on using ioControl commands (instructions) to program your strategy effectively, see Chapter 10, "Programming with Commands," and the individual command information in the *ioControl Command Reference*.

In this Chapter

Steps to Design

How do you get from your real-world control problem to a working ioControl strategy that solves it? Here's an outline of the basic approach; we'll fill in the details on the following pages.

First, solve the problem.

- Define the problem.
  - What am I trying to do?
  - What inputs and data do I have to work with?
  - What do the outputs have to be?
  - How many times does the process have to be repeated?
- Design a logical sequence of steps to solve the problem.
  - Break down the larger process task into sub-process tasks.
  - Break down the sub-process tasks into detailed steps.
- Test the steps.

Next, build the strategy.

- Configure hardware.
  - Control Engines
  - I/O units
  - I/O points
- Determine necessary variables and configure them.
- Create charts, one for each sub-process, and add instructions.
- Compile and debug each chart.
- Compile and debug the whole strategy.

Finally, use and improve the strategy.

Now let's take a look at each of these steps in order.

Solving the Problem

You can avoid a lot of extra time and rework if you define and solve your problem before you ever start building a flowchart in ioControl.

Defining the Problem

Suppose, for example, you want to automate a simple lawn sprinkler system. Start by asking yourself (or others) questions about the control process you're trying to do.

What are you trying to do? Start with the big picture and work down to the smaller details.

- Big picture: I'm trying to control this sprinkler system.
- Smaller details:
  - The sprinklers should turn on every Sunday and every Wednesday.
  - They should not turn on if it's raining.
  - They should start at six o'clock in the morning.
  - They need to run for 15 minutes.

What inputs and data do you have to work with? List all the inputs. Describe the data they provide. Check for any inputs or data you need but don't have.

Input	Data the input provides
Hygrometer	Humidity (percentage from 0–100%)
Day	Day of the week (Sunday through Saturday)
Time	Hour of the day (24-hour clock)

Input	Data the input provides
Sprinkler status	Whether sprinklers are currently running (on or off)
Input from timer	Length of time sprinklers have been on (in minutes)

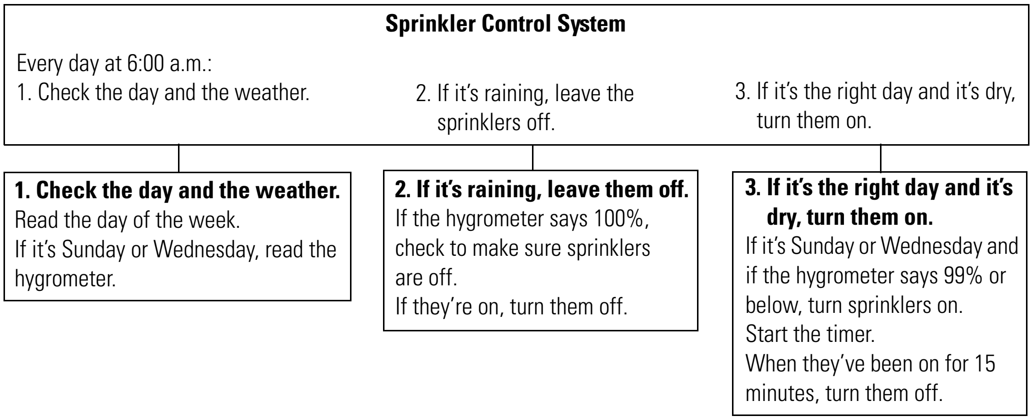
What do the outputs need to be? List all the outputs. Describe the results that are needed. Check for any outputs you need but don't have.

Output	What it does
Sprinkler switch	Turns sprinklers on or off
Timer control	Sets timer

How many times does the process have to be repeated? Our example is a simple sprinkler system in which the process happens twice a week on certain days. In a more complex system, however, you might have one section of the sprinklers turn on, followed by other sections, repeating the process several times in different areas and then repeating the whole pattern on certain days.

Designing a Logical Sequence of Steps to Solve the Problem

Now that you've determined what you're trying to do and what your inputs, data, and outputs look like, you're ready to outline the steps needed to solve the control problem. Think about how you would do the job if you were doing it by hand. Again, work from the top down, starting with big steps and then breaking those big steps into smaller steps.



Sprinkler Control System

Every day at 6:00 a.m.:
1. Check the day and the weather.

2. If it's raining, leave the sprinklers off.

3. If it's the right day and it's dry, turn them on.

1. Check the day and the weather.
Read the day of the week.
If it's Sunday or Wednesday, read the hygrometer.

2. If it's raining, leave them off.
If the hygrometer says 100%, check to make sure sprinklers are off.
If they're on, turn them off.

3. If it's the right day and it's dry, turn them on.
If it's Sunday or Wednesday and if the hygrometer says 99% or below, turn sprinklers on.
Start the timer.
When they've been on for 15 minutes, turn them off.

If a human being were doing this job, this level of instruction would probably be just fine. With a computer, however, the instructions need more detail. For example, the first instruction, "Every day at 6:00 a.m.," would have to be expanded more like this:

- **a.** Read the hour.
- **b.** If the hour equals six, go on to the next step.
- **c.** If the hour does not equal six, go back to step a.

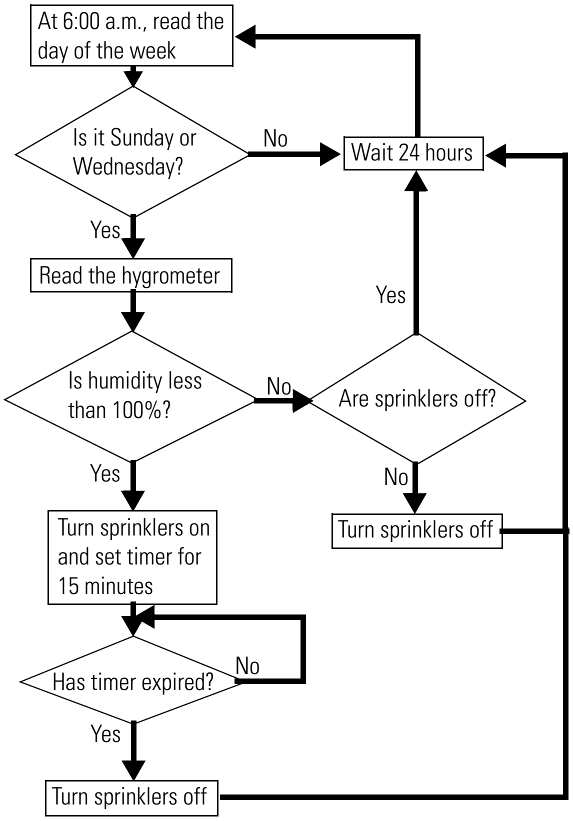
Testing the Steps

Now that you have your steps in place, run through each sub-process in your mind to see if anything is missing. Is there an answer for every "if"? Are all the possibilities taken into account?

It may help you to draw out your control process as a flowchart. Often it is easier to see decision points and responses to decisions in a flowchart. You're still working at the human level at this point; just be aware that the computer may require more details.

Here is a possible flowchart for the simple sprinkler control problem.

Simple Sprinkler System Process Flowchart


At 6:00 a.m., read the day of the week
Is it Sunday or Wednesday?
No
Wait 24 hours
Yes
Read the hygrometer
Yes
Is humidity less than 100%?
No
Are sprinklers off?
Yes
No
Turn sprinklers on and set timer for 15 minutes
Turn sprinklers off
Has timer expired?
No
Yes
Turn sprinklers off

Building the Strategy

Once you have the control problem solved in detail, you can begin building the strategy in ioControl. Now you'll add all the logical details the computer needs.

Configuring Hardware

The first step in building your strategy is to configure your control engine, I/O units, and I/O points. (For more information and step-by-step procedures, see Chapter 5, "Working with Control

Engines," and Chapter 6, "Working with I/O.") You can add more I/O units and points later if needed, but it saves time to configure them now.

When you solved this control problem, some of the inputs and outputs you defined were physical I/O, such as the hygrometer and the switch to turn on the sprinklers. You configure these physical inputs and outputs as the I/O points.

Here are the I/O points you might configure for the simple sprinkler system:

Physical I/O	Type	I/O Point Name
Hygrometer	Analog Input	Hygrometer
Sprinkler status	Digital Input	Sprinkler_Status
Sprinkler switch	Digital Output	Sprinkler_Switch

Determining and Configuring Variables

Some of the inputs and outputs you defined when you solved the problem were not physical I/O, but information. For example, the day of the week is not a physical input; it's a piece of information you can get using a command in ioControl. These inputs and outputs become variables.

You also need variables to store the information that input points give you. And you may need variables for manipulating information.

To determine the variables you need, think about the pieces of information your process requires. Then look at the ioControl commands in the *ioControl Command Reference*. Find the commands you need and check them to see what types of variables each command requires.

For example, you need to know what day of the week it is. Will the control engine give you this information as a string (for example, "Wednesday") or in some other format? When you check the command Get Day of Week, you discover that the day is returned as a number (Wednesday = 3), so you set up this variable as a numeric integer.

Here are some variables you may need. (See Chapter 9, "Using Variables and Commands," for more information and step-by-step procedures to add variables.) You can change them or add other variables later if necessary.

Variable Needed	Type	Name in ioControl
Hour of the day	Numeric (32-bit integer)	Time_of_Day
Day of the week	Numeric (32-bit integer)	Day_of_the_Week
Humidity	Numeric (float)	Humidity
Down timer	Timer	Timer

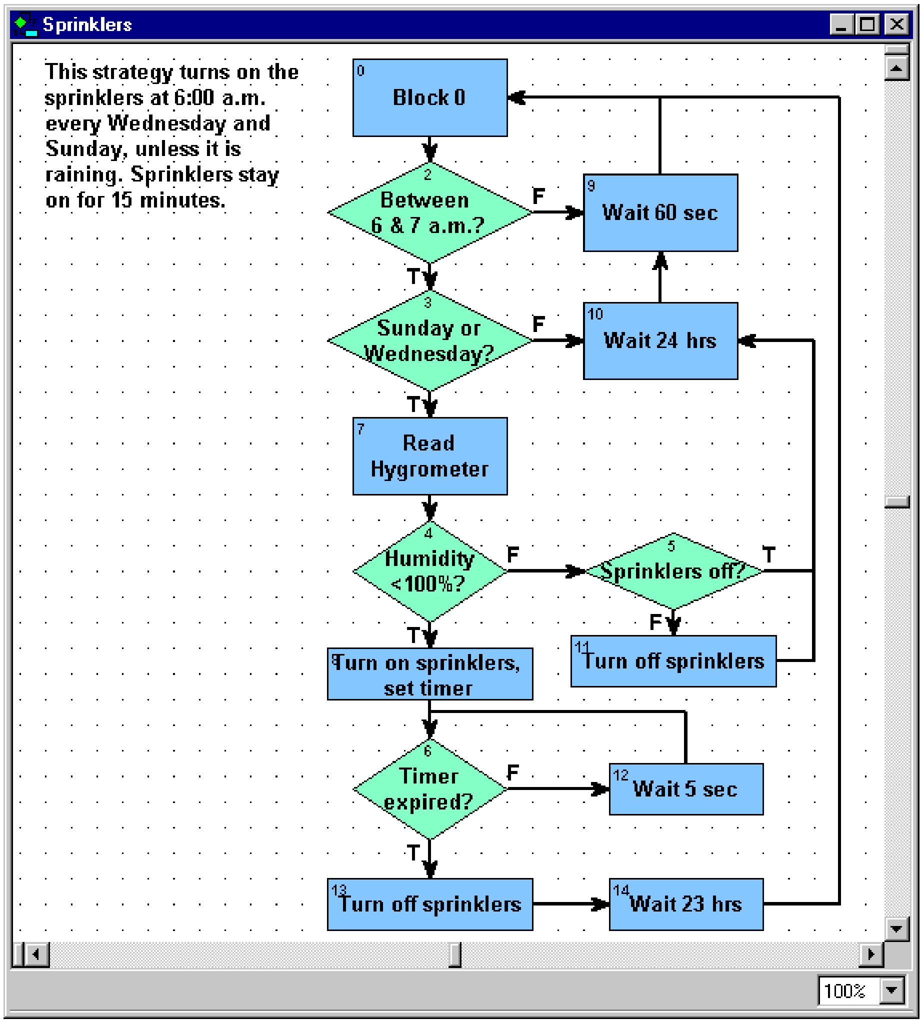
Creating ioControl Charts and Adding Instructions

If you have already drawn the process in a flowchart, this step will go faster. ioControl is based on flowcharts because they are a natural way to show a control process. And the block names and instructions you add to charts—just like the names of variables—are in normal, everyday language. (For more information and step-by-step procedures, see Chapter 8, "Working with Flowcharts," and Chapter 9, "Using Variables and Commands.")

The difference between the flowchart you drew before and the chart you create in ioControl is that the flowchart was designed for human beings, but the ioControl chart must be more detailed so the computer can follow it.

Because the chart is more detailed—and because most control processes are far more complex than our sprinkler system—you will usually create multiple charts for an ioControl strategy. Break the process down into logical chunks, or modules, and then create a chart for each module. A modular design makes it easier to test and to update your strategy.

Even for a simple process like our sprinkler control, there is no single "correct" way to solve the control problem and build the strategy. Instead, there are many possible ways. The following chart shows one example:

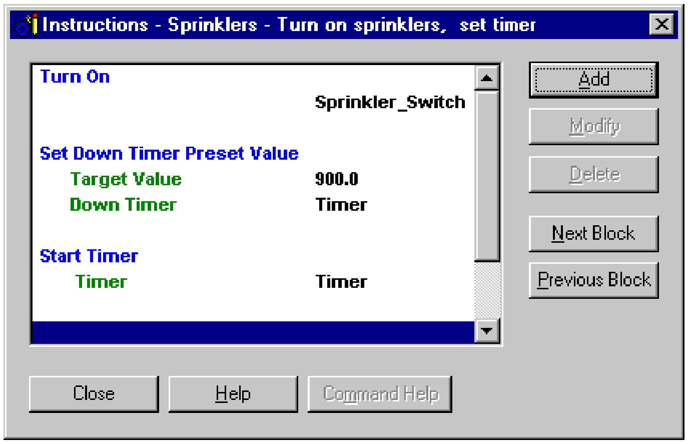


This strategy turns on the sprinklers at 6:00 a.m. every Wednesday and Sunday, unless it is raining. Sprinklers stay on for 15 minutes.

Notice one of the differences between this chart and the human-level chart on page 4-4: several delays have been added to the chart. There is one before rechecking the hour of the day, one before rechecking the day of the week, one before rechecking whether the timer has expired, and finally one before starting the flowchart over again.

This sprinkler system is not time-critical. If the sprinklers turn on at a little past 6:00 a.m. or the grass is watered an extra few seconds, it doesn't matter. Delays give the control engine time to do other things. (For more information on delays and using control engine time effectively, see "Optimizing Throughput" on page 4-21.)

Each block in the chart contains the ioControl commands (instructions) that do the work in the system. For example, here are the instructions for the block "Turn on sprinklers, set timer":



Instructions - Sprinklers - Turn on sprinklers, set timer

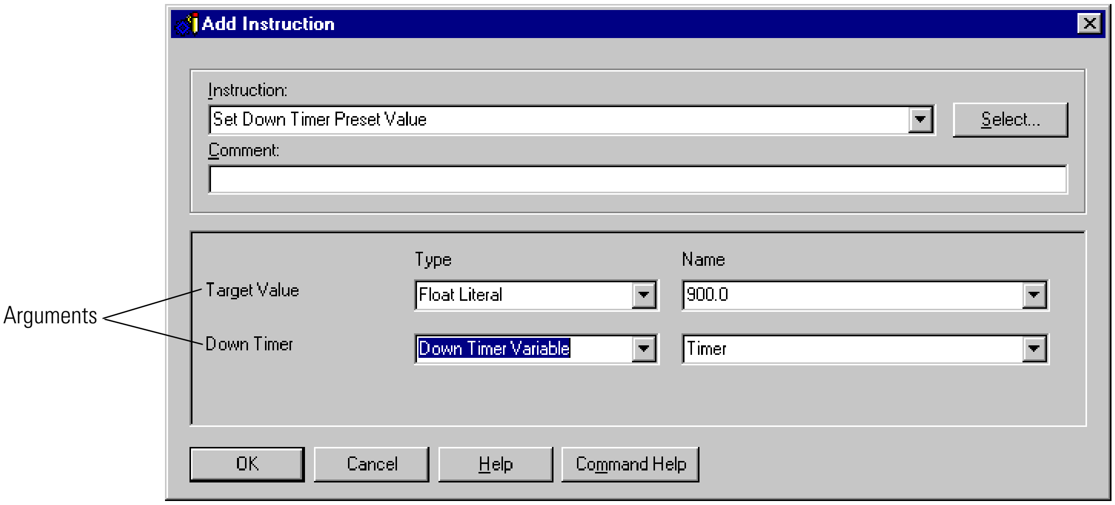
Turn On
Sprinkler_Switch

Set Down Timer Preset Value
Target Value 900.0
Down Timer Timer

Start Timer
Timer Timer

As you create charts and enter instructions, keep referring to the *ioControl Command Reference*. The *Command Reference* describes the "arguments" for each command, which are pieces of information you must enter in the instruction. You can also press F1 to open online help for commands, which gives you the same information as in the printed *Command Reference*.

For example, one of the commands shown above, Set Down Timer Preset Value, has two arguments. The *Command Reference* and online help tell you that the first argument—the target value, or the value from which the timer counts down—can be a float variable or a float literal, and that the second argument is the name of the down timer variable. You need this information when you enter the instruction in the Add Instruction dialog box.



Instruction: Set Down Timer Preset Value

Comment:

	Type	Name
Target Value	Float Literal	900.0
Down Timer	Down Timer Variable	Timer

The sample flowchart we've shown uses standard ioControl commands only. If you have a programming background, you may wish to incorporate OptoScript blocks in your flowchart. OptoScript is a procedural language that can simplify some programming tasks, including string handling, math calculations, and complex loops and conditions. See Chapter 11, "Using OptoScript," for more information.

Compiling and Debugging the Strategy

When all charts are completed and their instructions added, the next step is to compile and debug the strategy. But first, check your hardware. Check cabling and connections, and make sure the actual I/O matches the configured I/O in the strategy.

Now compile and debug the strategy. (See Chapter 7, "Working with Strategies," for more information and step-by-step procedures. If you have problems, see Appendix A, "ioControl Troubleshooting,") If you have multiple charts, debug each one separately and then debug the strategy as a whole. It is easier to find errors in one flowchart than in a group of interrelated ones.

Make sure the connections between charts are accurate. For example, is a chart started at the correct block in another chart?

Use the debugging tools discussed in Chapter 6 to find errors by stepping through a chart and setting breakpoints to discover which block or line contains the problem. And if you've tried everything and it still doesn't work, contact Opto 22 Product Support. (See page 1-4.)

Using and Improving the Strategy

Any strategy is one of several possible ways to solve a control problem. One way may be more efficient under some circumstances; another way may be better for others. As you use the strategy over time, as control needs change, and as you become more knowledgeable about ioControl, you'll find ways to make the strategy better.

Basic Rules

The sprinkler strategy we just created is a simple example. This section gives basic rules to keep in mind when you're solving more complex control problems.

When you create a new ioControl strategy, a Powerup chart is automatically included. You add all the other charts you need, and the total number of charts in the strategy is limited only by the control engine's memory. Be aware, however, that the maximum number of charts *that can be running at any one time* is based on the control engine you are using:

- 32 charts on a SNAP PAC S-series controller
- 16 charts on a SNAP PAC R-series or SNAP-LCE controller
- 8 charts on SNAP Ultimate I/O

Program logic moves through a flowchart in one of two ways: flow-through or loop.

- A chart with **flow-through logic** performs a set of specific commands and then stops. It has a beginning and an end, and at the end is a condition or action block that has no exit.

  Subroutines, the Powerup chart, and any other chart that does not need to run continuously should always have flow-through logic. In complex strategies, be careful when using delays and condition looping (waiting for something to occur) in charts with flow-through logic.

- A chart with **loop logic** has no end. It loops through a set of actions and conditions continuously. A loop-logic chart can have several paths through which the direction of the logic may flow, depending on certain criteria. Use loop logic for a chart that needs to run continuously. (The simple sprinkler chart has loop logic; it runs continuously.)

Chart Guidelines

As you design your strategy, put each sub-process of your overall control problem into a separate chart within the strategy. As noted above, on a SNAP Ultimate brain, you can have a maximum of eight charts (plus one host task) running at once. On a SNAP PAC R-series or SNAP-LCE controller, a maximum of 16 charts (plus one host task) can run at once. On a SNAP PAC S-series controller, 32 charts (plus one host task) can run at once. (For more information, see "Optimizing Throughput" on page 4-21.)

In general, follow these chart guidelines:

- Use the Powerup chart just to set initial values for variables, perform setup commands, and start the main charts. Use flow-through logic so the Powerup chart will stop as soon as the other charts begin.
- Create a few charts to monitor essential or time-critical pieces of your process, such as emergency stops on dangerous equipment or I/O that must be monitored continuously. These charts should use loop logic so they are constantly running.
- If a set of operations is used more than once in your strategy or is used in more than one strategy, you can put it in a subroutine and call the subroutine when needed. Calling a subroutine doesn't require as much time as calling a chart or starting a new chart. In addition, a subroutine starts as soon as it is called, but a chart is simply placed in the task queue and started in turn. See Chapter 12, "Using Subroutines," for more information.
- Use the text tool to type a descriptive title in each chart and add any necessary explanations. Keep blocks far enough apart to easily see the flow, and if possible, design the chart so its entire flow can be seen within one window. If a chart becomes too complex, split it into smaller charts.
- When you use OptoScript code, remember that it is not self-documenting. Be sure to add comments so that you or others can easily see what the code is doing. (Similar comments are also useful in other ioControl flowchart blocks.) Use OptoScript blocks within a flowchart for operations they make easier, such as string handling and math calculations, but keep the logic and purpose of the strategy clear in the flowchart.
- Within a chart, use the start block (Block 0) for setting initial values but not for control. If a block contains more instructions than can be easily traced and debugged, break it down into two or more sequential blocks. Never place another block in the flowchart before Block 0.

Naming Conventions

To save trouble in the long run, it's wise to establish naming conventions for charts, blocks, variables, I/O units and points, control engines, subroutines, and so on. If you name these elements in a consistent way from the beginning, it is easier to find them in lists, to see related elements, and to know their functions without opening them.

Since ioControl automatically alphabetizes these names, you can plan naming schemes to place similar items together in the way you expect to use them.

Names have a maximum length of 50 characters. They can be all upper-case characters, or they can be mixed case.

In chart names, for example, you might include a few letters indicating whether the chart monitors critical tasks, is a master chart calling others, or is periodically called:

- Mntr_Tank_Leak (constantly running chart that monitors a critical situation)
- Mstr_Conveyor_Process (master chart that calls others for a sub-process)
- Call_Message_Display (chart called by a master chart for a specific purpose)

Block names should tell what the block does, so you can understand a chart without needing additional explanations. Block names should describe the purpose of the instructions within them. Use a question for the name of a condition block, with a true exit reflecting the answer yes.

- Read Thermometer (action block)
- Temperature Too High? (condition block)
- Turn On Pump 1 *or* Pump 1 On (action block)
- Pump On? (condition block)

Variable names can use Hungarian notation to indicate the type of variable, since the variable type is not always apparent in its name. See "Variable Name Conventions" on page 11-14 for a suggested list of notations.

Variable names might also include a way to distinguish variables used in a single chart or process from variables used by more than one chart or process. If your strategy is large, it's helpful to separate them. Variables that are used only in a table could include the table name in their name, for example Fail_Count_in_Config_Values. (Config_Values is the table name.)

In I/O point names, you may want to indicate the point's function, the state of the device when the point is active, its location, or a schematic reference. You can abbreviate names of familiar devices and write out less familiar names. Include the information you need in the order in which it will be most useful to you:

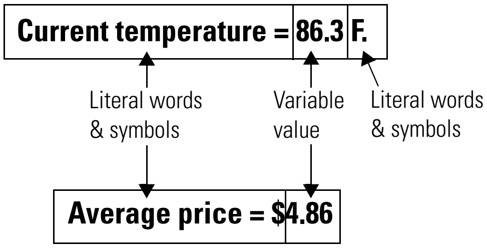
- Heater_Switch (You have only one heater.)
- Htr6_Switch_SW23B (You have many heaters, and the schematic reference is needed.)
- Cnvyr_Speed_Encoder_BldgA (You want all conveyors to appear together.)
- BldgA_Cnvyr_Speed_Encoder (You want all points in Building A to appear together.)

Instruction Examples

This section includes examples of common instructions you may want to use. See Chapter 10, "Programming with Commands," for additional information on programming with ioControl. If you need to use math calculations, strings, or complex loops and conditions in your strategy, also see Chapter 11, "Using OptoScript," for examples of OptoScript. OptoScript is a procedural language within ioControl that can make programming easier, especially if you are an experienced programmer.

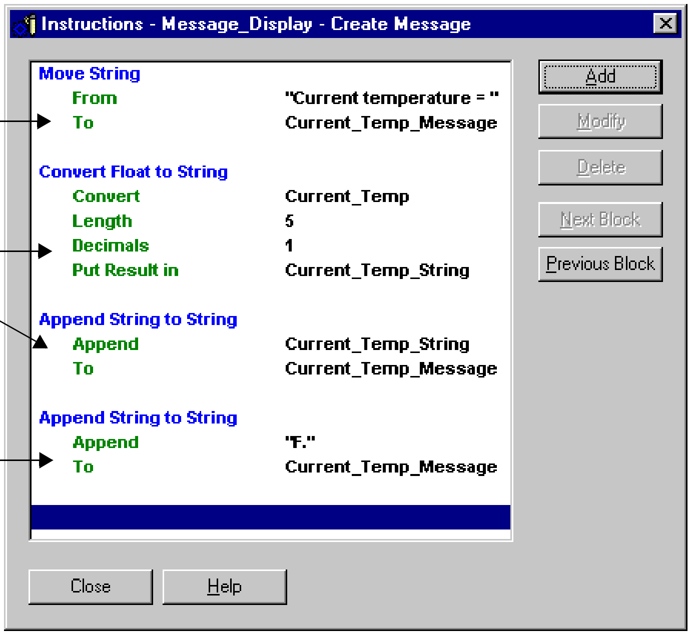
Creating Messages to Display On Screen

You may need to create messages to display on screen, for example to give data to operators. Typically these messages consist of some specific, literal words and symbols followed by a variable value and maybe some additional literal words or symbols.



You can create the message in a single block in your ioControl chart, like this:

1. To enter the literal text, use the command Move String. Remember to include spaces where you want them to appear, such as after the equal sign.
2. Add the variable value by converting it to a string and appending it to the string you just created.
3. If needed, append another literal string.



```
Move String
    From                  "Current temperature = "
    To                    Current_Temp_Message

Convert Float to String
    Convert               Current_Temp
    Length                5
    Decimals              1
    Put Result in         Current_Temp_String

Append String to String
    Append                Current_Temp_String
    To                    Current_Temp_Message

Append String to String
    Append                "F."
    To                    Current_Temp_Message
```

Alternatively, you can create the message by placing the following OptoScript code within a single OptoScript block:

```
FloatToString(Current_Temp, 5, 1, Current_Temp_String);
Current_Temp_Message = "Current temperature = " + Current_Temp_String +
"F.";
```

OptoScript can be more efficient for string handling than standard ioControl commands. See Chapter 11, “Using OptoScript,” for more information.

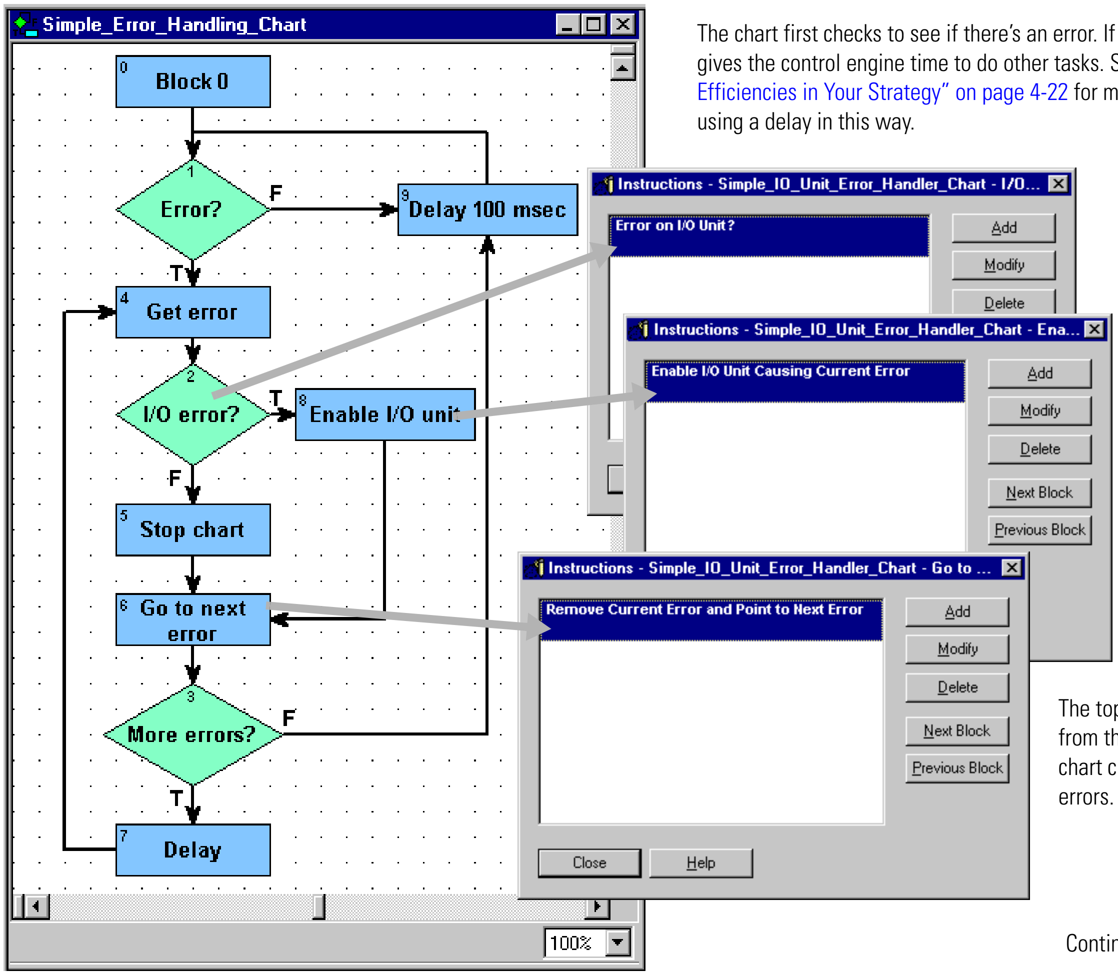
Error Handling

Every strategy should have a way to handle errors. It's important to check values and errors that are returned from the commands in your strategy. An I/O unit, for example, will be automatically disabled if a command sends it variable values that are clearly wrong, such as a memory map address in an incorrect format.

You can easily build a simple error handling chart like the one shown below, which automatically checks errors and other messages and takes action based on the type of error.

This chart, for example, checks for I/O unit errors. Suppose a remote I/O unit loses power for a few seconds, and then the power comes back on. When the power is lost, communication to the unit is disabled. To enable it again, you could stop the strategy and restart it, or you could enable the unit in the debugger. With this error handling chart, however, you can automatically enable the I/O unit and restore communication while the strategy is running.

The chart checks the top error in the message queue. If it is an I/O unit error, communication to the unit is enabled. If it is not an I/O unit error, another action happens (another chart in the strategy is stopped). Two delays are built in: the first one to release the time slice if an error is not found, and the second one to allow the control engine time to restore communication with the I/O unit.



Instructions - Simple_IO_Unit_Error_Handler_Chart - I/O...

Error on I/O Unit?

Instructions - Simple_IO_Unit_Error_Handler_Chart - Ena...

Enable I/O Unit Causing Current Error

Instructions - Simple_IO_Unit_Error_Handler_Chart - Go to ...

Remove Current Error and Point to Next Error

The chart first checks to see if there's an error. If not, a slight delay gives the control engine time to do other tasks. See "Increasing Efficiencies in Your Strategy" on page 4-22 for more information on using a delay in this way.

If the error is an I/O unit error, the I/O unit that caused it is enabled (communication to that unit is restored).

The top error is removed from the queue, and the chart checks for more errors.

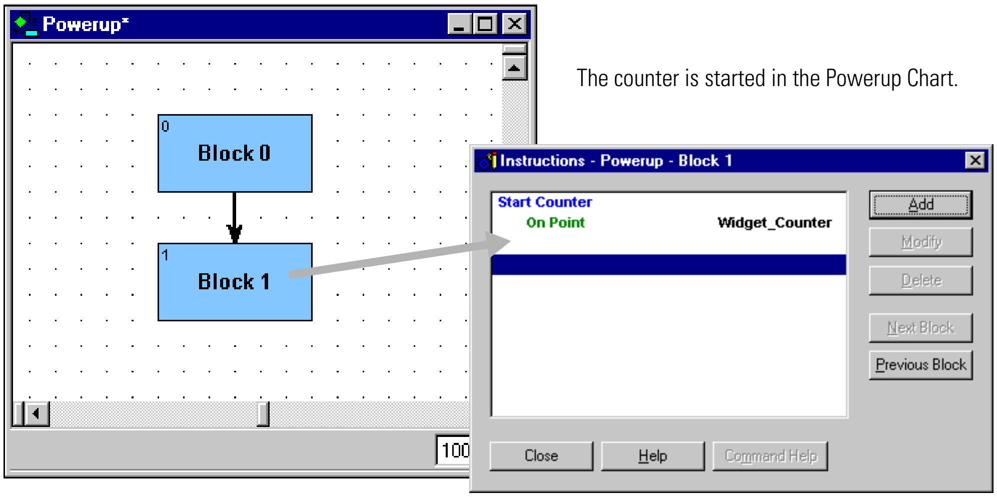
Continued on next page

The error chart shown on the previous page is designed to be started by the Powerup chart and to run continuously. If necessary, you might modify the chart to take different actions depending on the error code, using the command Get Error Code of Current Error. Or you might want to track problem I/O units with the command Get Name of I/O Unit Causing Current Error, and then save the I/O unit names to a string table. These and other commands used in error handling are listed on page 10-56.

Counting

If an I/O unit supports counting, any digital input on it can be used as a counter. Counters count the number of times the input changes from off to on. **You must start the counter before you can read it.** See "Counters" on page 10-3 for more information.

Counters are often started in the Powerup Chart. For example, suppose you wanted to count the number of widgets moving along a conveyor belt from manufacturing to shipping. You could start the counter in the Powerup Chart, like this:



Start Counter
On Point Widget_Counter

The counter is started in the Powerup Chart.

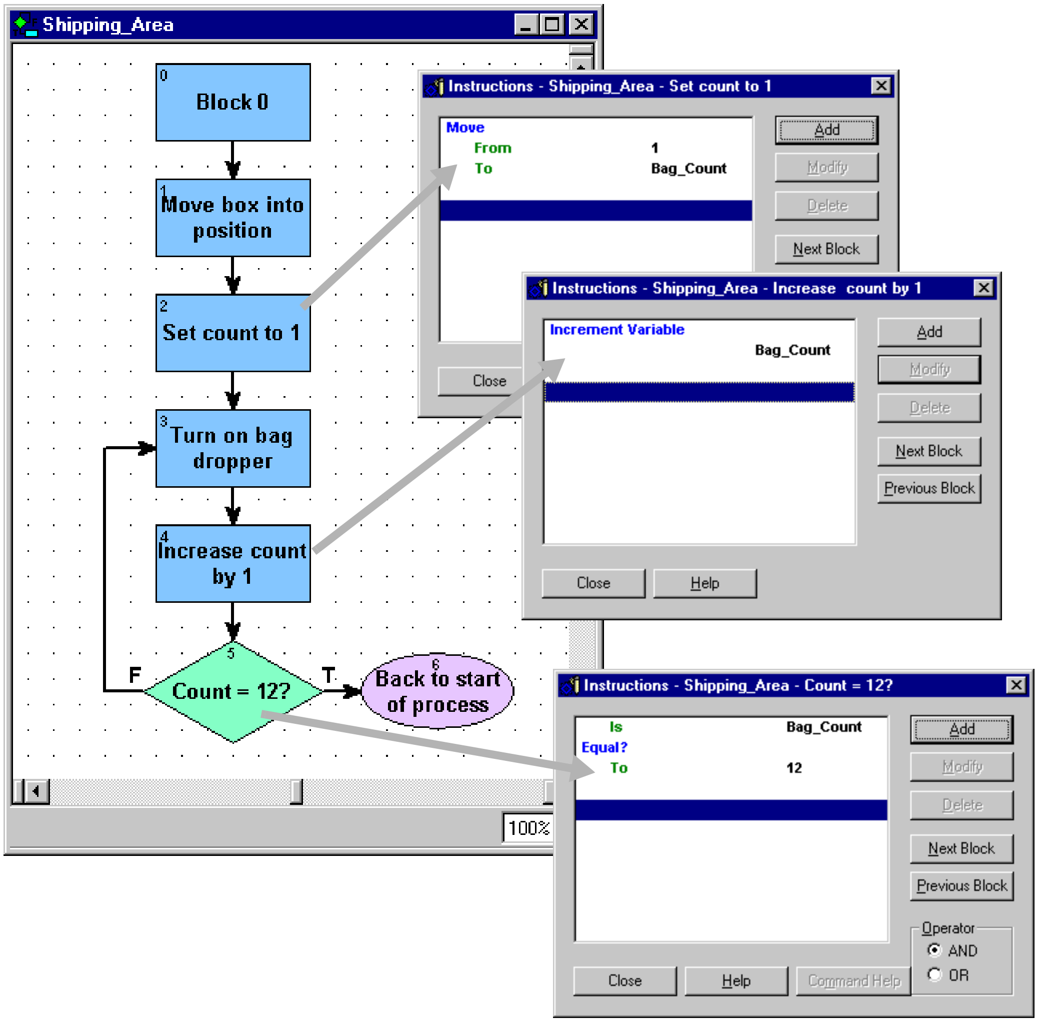
Once the counter is started, you can read its value at any time by using the command Get Counter.

If you want the counter to start over when you read its value—for example, to count the number of widgets in each 24-hour period—use the command Get & Clear Counter.

Using a Count Variable for Repetitive Actions

A numeric variable for counting is useful when the same action needs to be repeated a specific number of times. For example, suppose your process includes filling a packing box with a dozen bags of cookies. You can use a count variable to track how many bags are dropped into the box.

Here's part of a chart showing how you would use a count variable in this way:



Instructions - Shipping_Area - Set count to 1

Move
From 1
To Bag_Count

Instructions - Shipping_Area - Increase count by 1

Increment Variable Bag_Count

Instructions - Shipping_Area - Count = 12?

Is Bag_Count
Equal?
To 12

If you are an experienced programmer, you'll notice that this example is a `for` loop. You can use the following OptoScript code within a single OptoScript block to accomplish the same result:

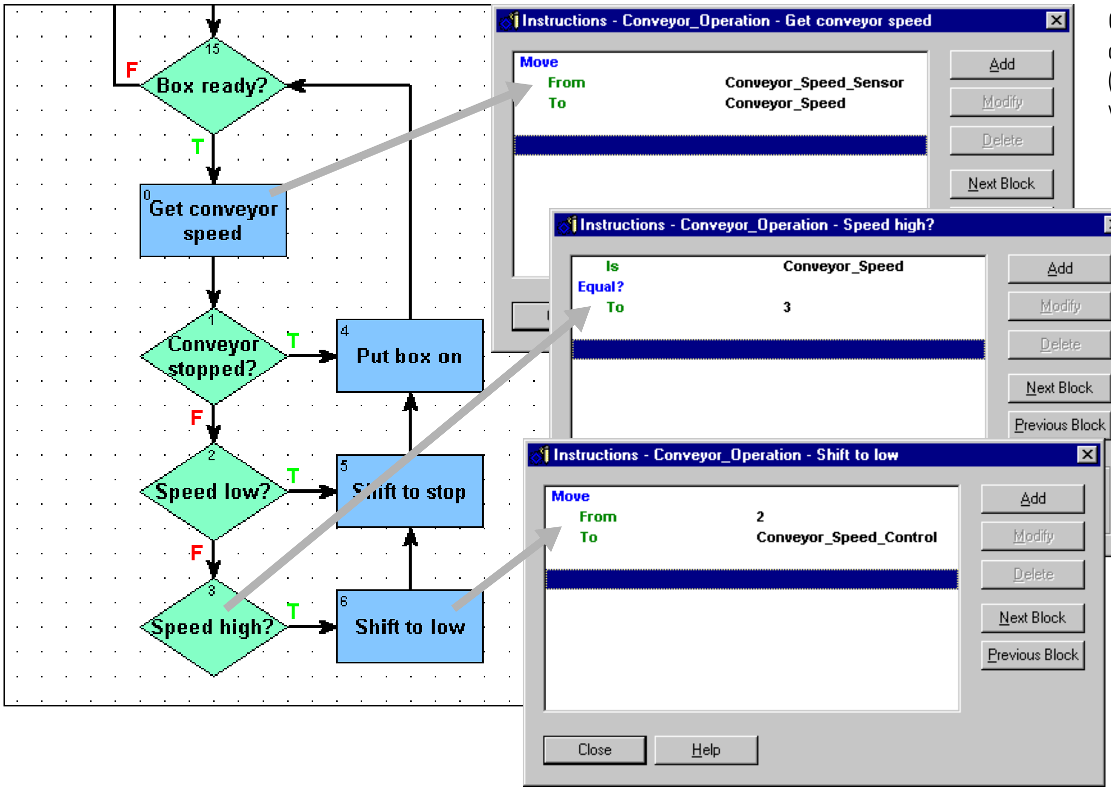
```
for Bag_Count = 1 to 12 step 1 //Count 12 items dropped into box
  Bag_Dropper = 1; //Turn on bag dropper
next
```

See Chapter 11, "Using OptoScript," for more information on using code to streamline control structures in your strategy.

Programming Case Statements

A frequent need in programming is to create case statements, also called “switch” statements, “if/then/else” statements, or “nested if” statements. These statements create multiple decision points: if the condition is A, then X will happen; if the condition is B, then Y will happen, and so on. (This example uses regular ioControl commands; see Chapter 11, “Using OptoScript,” for another option that takes up less space in your flowchart.)

For example, suppose you have a conveyor with three speeds: high (3), low (2), and stopped (1). Sometimes it runs at high speed, sometimes at low speed; but before a box can be put on the conveyor, the conveyor must be stopped. And it can only be stopped from low speed, not from high speed. Here’s a portion of a chart showing the case statements that control this process:



Instructions - Conveyor_Operation - Get conveyor speed

Move
From Conveyor_Speed_Sensor
To Conveyor_Speed

Instructions - Conveyor_Operation - Speed high?

Is Conveyor_Speed
Equal?
To 3

Instructions - Conveyor_Operation - Shift to low

Move
From 2
To Conveyor_Speed_Control

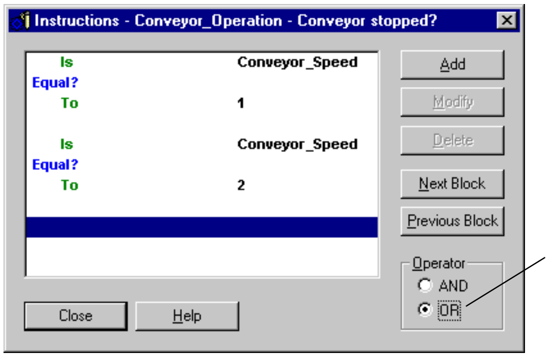
Conveyor speed is checked and moved (copied) into a variable.

Each condition block checks for a particular speed. This one checks for high speed (3).

If speed is too high, it is shifted down to the next level, until the conveyor is finally stopped and the box can be put on.

The example above shows three cases, because there are three possible speeds and each speed demands a different action.

If you had only two possibilities, for example if the box could also be put on the conveyor at low speed, you could handle both possibilities within one condition block. For example, you could put two Equal? commands in the same condition block, and check the Or operator, as shown at right.



Instructions - Conveyor_Operation - Conveyor stopped?

Is Conveyor_Speed
Equal?
To 1

Is Conveyor_Speed
Equal?
To 2

Operator
○ AND
● OR

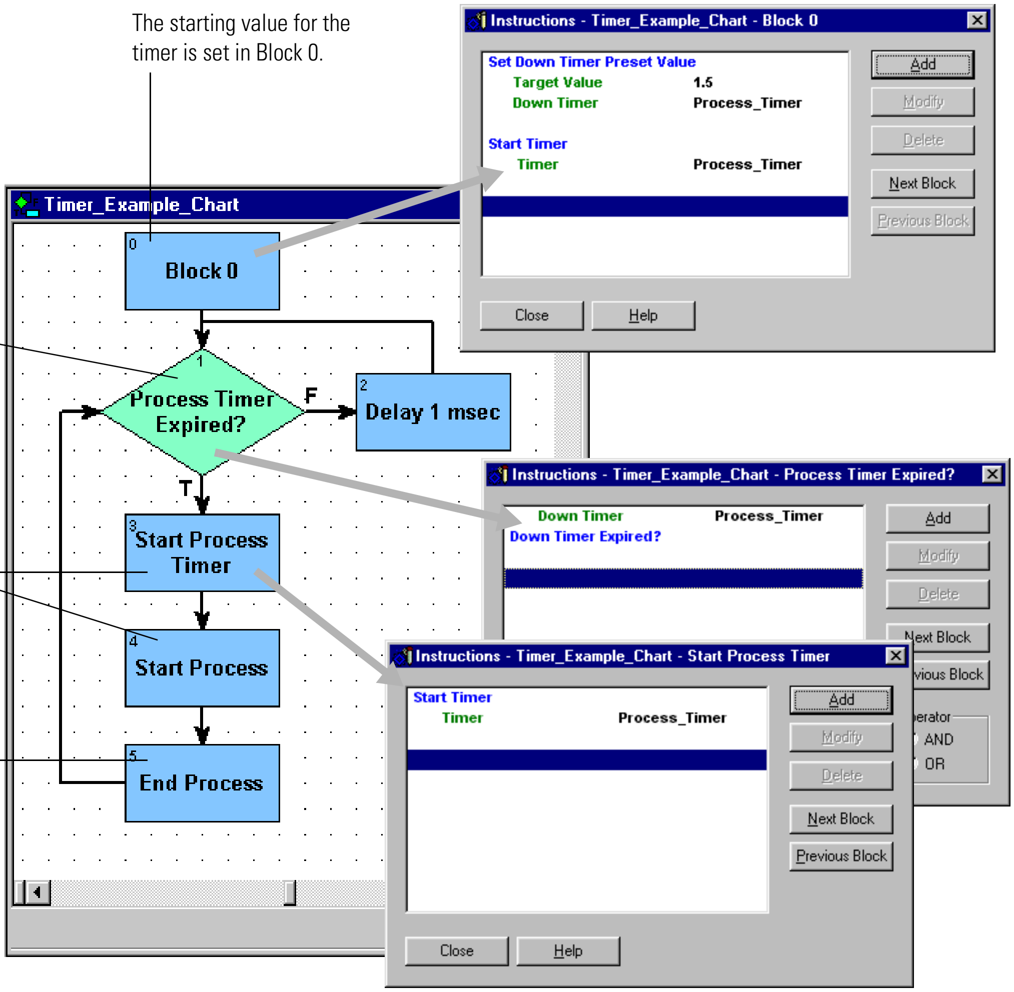
Or operator

Using a Timer

Timers come in two forms:

- Up timers count up from zero and are generally used to measure how long something takes.
- Down timers count down to zero from a number you set and are frequently used to determine when events should begin.

Here's an example of a down timer. This process starts every 1.5 seconds.



Instructions - Timer_Example_Chart - Block 0

Set Down Timer Preset Value
Target Value 1.5
Down Timer Process_Timer

Start Timer
Timer Process_Timer

Instructions - Timer_Example_Chart - Process Timer Expired?

Down Timer Process_Timer
Down Timer Expired?

Instructions - Timer_Example_Chart - Start Process Timer

Start Timer
Timer Process_Timer

The starting value for the timer is set in Block 0.

The condition block checks whether the timer has reached zero. If it hasn't, the chart loops until it has.

When the timer has expired, it is restarted and the process begins.

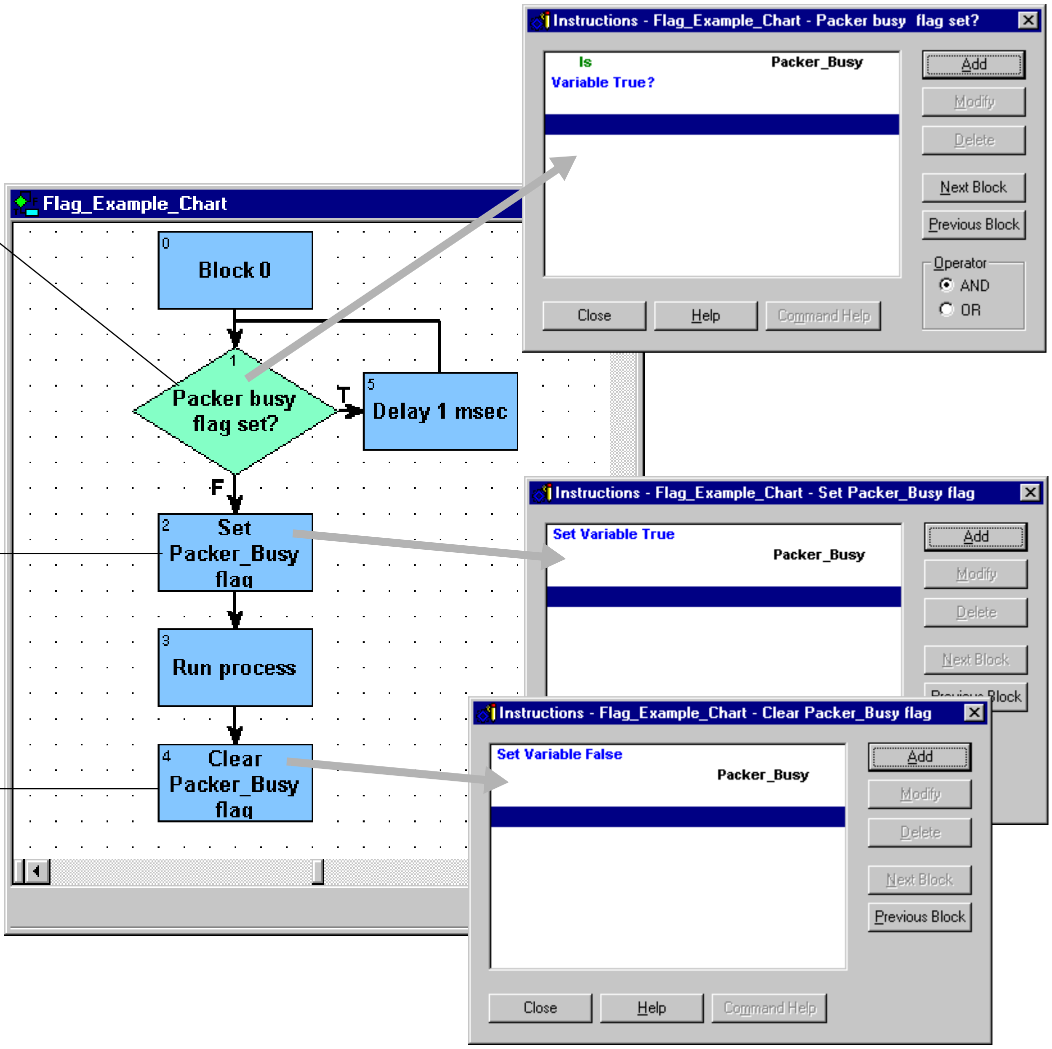
When the process ends, the logic loops back to check whether the timer has expired again.

Timers can be tricky. For additional details, see "Using Timers" on page 10-19.

Using a Flag

A flag is a way of controlling logical flow within a strategy. You can set a flag in one part of the strategy and then test it in another part. If the flag is set, the logic flows one way; if the flag is not set, it flows another way.

For example, a flag could be set to indicate that a machine is busy with one process, to prevent another process from using the machine at the same time. The following chart shows logic for one of the processes that uses the machine. If another process has already set the flag, this process must wait until the flag is cleared.



Instructions - Flag_Example_Chart - Packer busy flag set?

Is Packer_Busy
Variable True?

Operator: ● AND ○ OR

Instructions - Flag_Example_Chart - Set Packer_Busy flag

Set Variable True
Packer_Busy

Instructions - Flag_Example_Chart - Clear Packer_Busy flag

Set Variable False
Packer_Busy

The condition block checks whether the flag has been set, indicating that another process is using the machine. If the flag has been set, the chart logic loops until it has been cleared.

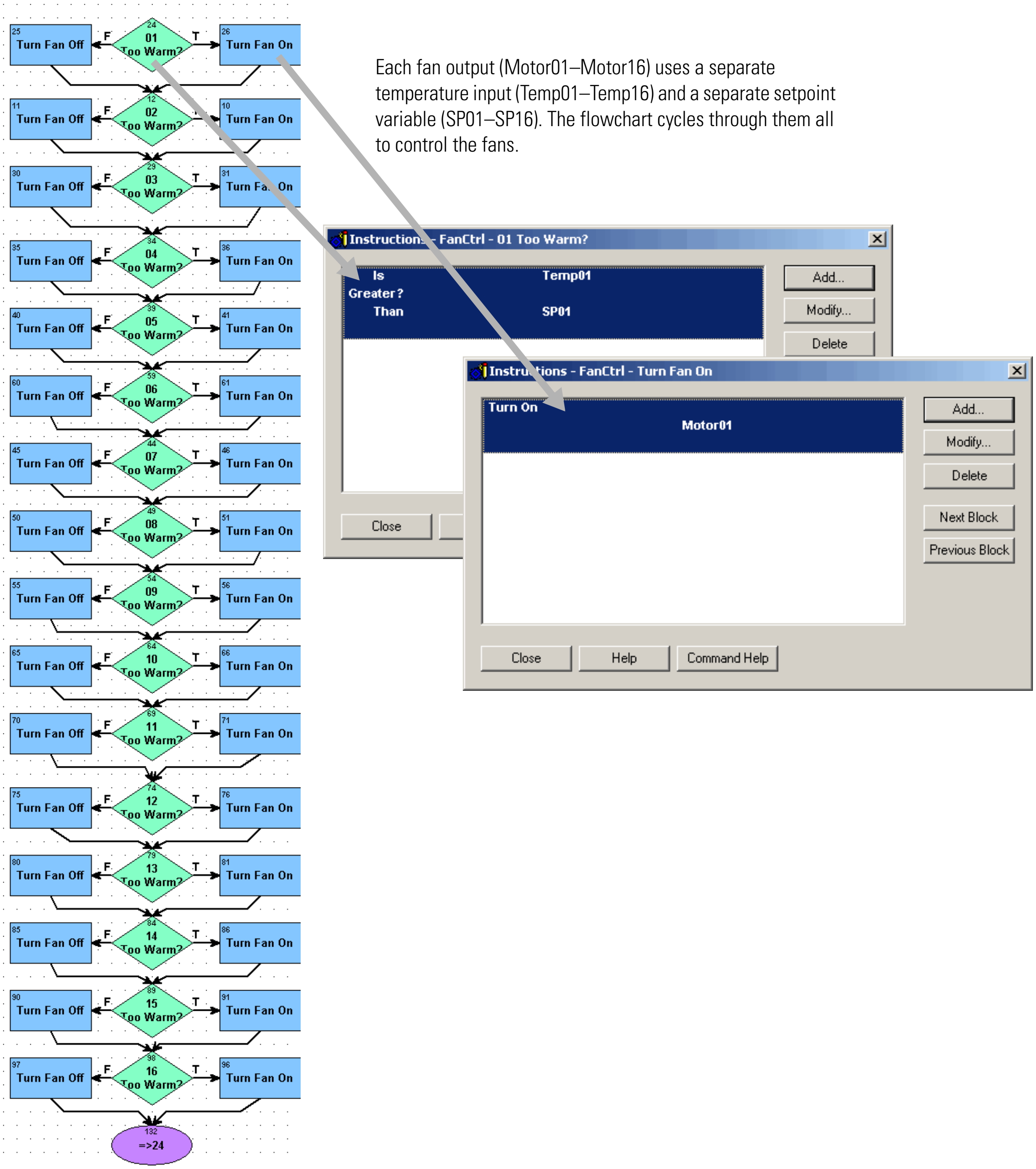
This block sets the flag so another process can see that the machine is busy.

When this process is complete, this block clears the flag, indicating that the machine is now available for use.

Pointers and Indexing

This example shows three possible ways to control 16 fans based on setpoints.

Without Pointers. The first way, shown below, uses individual fan outputs and setpoint variables to determine whether to turn off or on the appropriate fan. It's a simple way to solve the problem but takes a long time to program and produces a large, repetitive section in the flowchart. It also consumes more control engine memory, due to the large number of conditional and action blocks.



Each fan output (Motor01–Motor16) uses a separate temperature input (Temp01–Temp16) and a separate setpoint variable (SP01–SP16). The flowchart cycles through them all to control the fans.

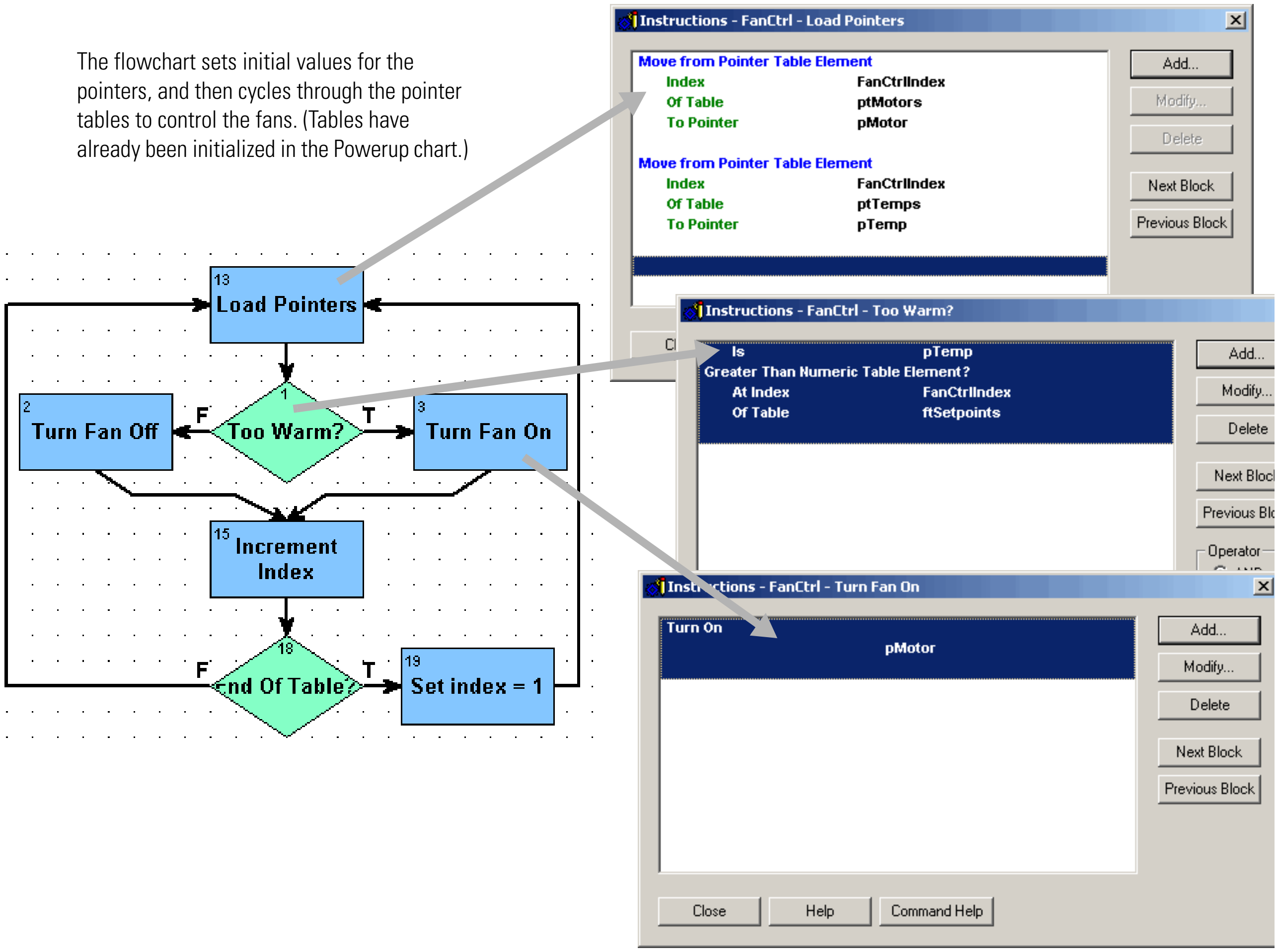
Instructions - FanCtrl - 01 Too Warm?

Is Greater? Than — Temp01 — SP01

Instructions - FanCtrl - Turn Fan On

Turn On — Motor01

With Pointers and Indexing. The second way uses pointers and indexing to accomplish the same result in a smaller space, using less control engine memory and fewer programming steps. The fan outputs and the temperature inputs are referred to by pointers. The setpoints, instead of requiring individual variables, are simply numbers in a table.



The flowchart sets initial values for the pointers, and then cycles through the pointer tables to control the fans. (Tables have already been initialized in the Powerup chart.)

Instructions - FanCtrl - Load Pointers

```
Move from Pointer Table Element
    Index           FanCtrlIndex
    Of Table        ptMotors
    To Pointer      pMotor

Move from Pointer Table Element
    Index           FanCtrlIndex
    Of Table        ptTemps
    To Pointer      pTemp
```

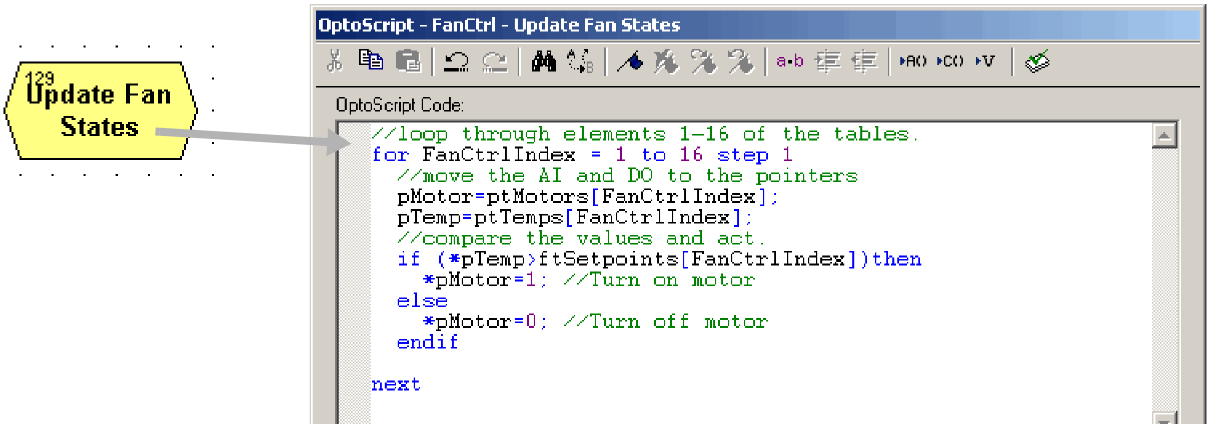
Instructions - FanCtrl - Too Warm?

```
    Is                      pTemp
Greater Than Numeric Table Element?
    At Index                FanCtrlIndex
    Of Table                ftSetpoints
```

Instructions - FanCtrl - Turn Fan On

```
Turn On
                pMotor
```

Using OptoScript. The third way also uses pointers and indexing, but places all the action in an OptoScript block. See Chapter 11, “Using OptoScript,” for more information on using OptoScript.



```
//loop through elements 1-16 of the tables.
for FanCtrlIndex = 1 to 16 step 1
  //move the AI and DO to the pointers
  pMotor=ptMotors[FanCtrlIndex];
  pTemp=ptTemps[FanCtrlIndex];
  //compare the values and act.
  if (*pTemp>ftSetpoints[FanCtrlIndex])then
    *pMotor=1; //Turn on motor
  else
    *pMotor=0; //Turn off motor
  endif

next
```

Optimizing Throughput

See additional related information on throughput in Opto 22 form #1302, the *ioDisplay User's Guide*.

Throughput can refer to communications between a PC and the control engine, or communications between the control engine and I/O. The following factors affect throughput for these two types of communication:

PC <-> Control Engine	Control Engine <-> I/O
Increasing host task frequency Using design efficiencies	Using I/O unit commands Using an I/O error-handling chart

In both types of communication, throughput is affected by control strategy design. The following sections discuss how to design your control strategy to maximize throughput.

Understanding ioControl Multitasking

When SNAP controllers run ioControl strategies, they can multitask, or run several tasks at once. The total number depends upon the controller you use:

- A SNAP PAC S-series controller can run up to 33 tasks (32 flowcharts plus the host task).
- A SNAP PAC R-series or SNAP-LCE controller can run up to 17 tasks (16 charts plus the host task).
- A SNAP Ultimate brain can run up to 9 tasks at once (8 charts plus the host task).

Total tasks include:

- Host task, which is a vehicle for communicating with a PC
- Powerup Chart
- Any other chart or subroutine in your strategy that is running or suspended. (A subroutine assumes the time slice of the chart that called it.)

The tasks are not actually run simultaneously; each task gets a small amount of time before the control engine moves to the next task in the queue. If a task is finished before its time is up, the control engine moves to the next task sooner. For example, a suspended chart uses no time, even though it is counted as a task.

Host Task

The host task functions as a slave, which means it never originates messages, but only responds to inquiries or commands. The host task runs by default, but it only uses time when there is communication to the controller.

The host task must be used to download new firmware to the control engine.

Optimizing PC to Control Engine Throughput

You can optimize throughput between the PC and the control engine by increasing the host task's frequency and by using design efficiencies in your strategy.

Increasing Host Task Frequency

If you minimize the number of tasks in the queue, the host task is done more frequently and all tasks get more time. To minimize tasks, combine processes so you have fewer charts. Here's how:

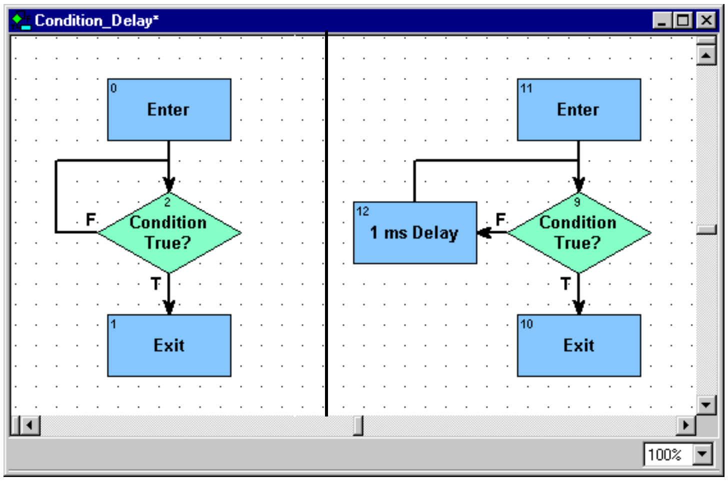
- Determine the processes that are time-critical. Put these processes in charts with looping logic that runs constantly.
- Use subroutines for any processes that recur within the strategy. A subroutine does not add an additional task; it assumes the time slice of the chart that called it. Make sure you check the status variable each time a subroutine is called to verify that the call was successful. See Chapter 12, "Using Subroutines," for more information.

Increasing Efficiencies in Your Strategy

A second way to optimize PC to Control Engine throughput is to increase efficiency in condition block loops.

If the condition block continually loops waiting for the condition to be true, the entire time slice for the chart is used up. However, you can build in a slight delay by using the command Delay (mSec) with a value of one. This command causes the chart to wait one millisecond before checking the condition again, and the rest of the chart's time slice is given up while it waits.

The following graphic shows two conditional loops. In the one on the left, the entire time slice is used up waiting for the condition to become true. In the example on the right, however, the delay allows the control engine to run other tasks while waiting for the condition to become true.



Ensuring Data Freshness for ioDisplay

Remember to develop your ioControl strategy in conjunction with your ioDisplay project to help optimize performance. To ensure maximum throughput, the strategy should be coordinated with Refresh groups in ioDisplay. For example, if ioDisplay is using typical freshness values of one second, then the strategy should read all of the I/O at least once every second. (One efficient way to do so is to use I/O Unit commands. See the section below.)

Optimizing Control Engine to I/O Throughput

Throughput between the control engine and I/O is also affected by strategy design. You can take advantage of I/O unit commands to communicate with several I/O points at once, and you can use an error-handling chart to handle I/O errors efficiently.

Using I/O Unit Commands

I/O unit commands speed up communications between the control engine and I/O by using tables to read or write all points on an I/O unit at once, rather than reading or writing one point at a time. For example, using the command Move I/O Unit to Numeric Table is many times faster than using the Move command once for each point. The command Move Numeric Table to I/O Unit is many times faster than using Turn On or Turn Off for each point.

Handling I/O Errors Efficiently

If the control engine encounters an error when communicating to I/O, it disables communication to the I/O unit. Disabling communication ensures that control engine performance is maintained. If an I/O timeout error occurred and communication with the I/O unit was not disabled, throughput to the remaining I/O units would drop significantly while the control engine tried to communicate.

If you use an I/O error-handling chart, make sure there is a reasonable delay after each attempt to re-enable communication to the I/O unit. In addition, for debugging purposes it is helpful if the error-handling chart logs the error. Investigate and correct the root cause of any I/O unit communication error to maintain throughput.

CHAPTER 5

Working with Control Engines

Introduction

This chapter shows you how to configure and work with control engines. See "Compatible Control Engines and I/O Units" on page 1-6.

In this Chapter

Configuring Control Engines

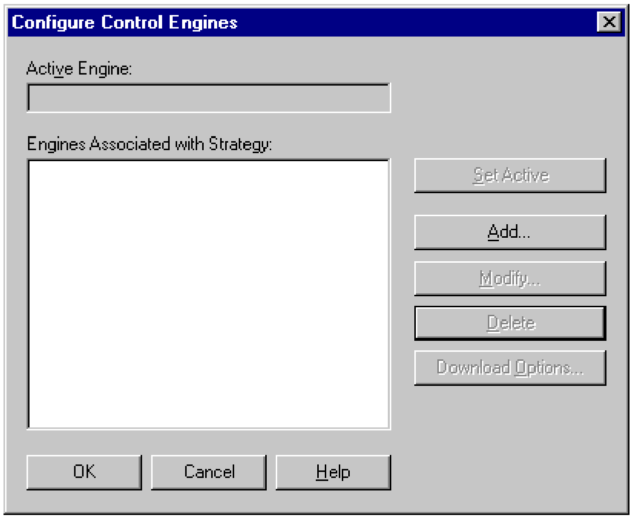
Before you can use a control engine to run a Strategy, you must first define the control engine on your PC and then associate the control engine with your ioControl strategy.

- See "Defining a Control Engine on Your PC" below to identify the connection through which the PC and the control engine communicate. Because this process writes to the Windows Registry on your PC, you must define control engines for each computer that uses your strategy. (If your computer can boot to two operating systems, you must configure control engines for each OS.) You can define control engines in ioControl or in the software utility ioTerminal.
- See "Associating the Control Engine with Your Strategy" on page 5-4 to identify which defined control engine is the active control engine. Although you can associate several control engines with the same strategy if necessary, the strategy can be downloaded to only one at a time. The control engine set to receive the download is called the active engine. You must use ioControl for associating the control engine with your strategy.

Defining a Control Engine on Your PC

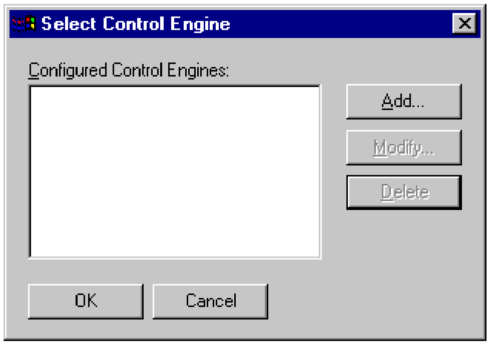
1. Choose one of the following:
   - **Using ioTerminal:** Choose Start→Programs→Opto 22→ioProject Software→Tools→ioTerminal. From the Configure menu, choose Control Engines to open the Select Control Engine dialog box. Skip to step 3.
   - **Using ioControl:** With a Strategy open in Configure mode or Online mode, double-click the Control Engines folder on the Strategy Tree.
     You can also click the Configure Control Engines button in the toolbar, select Configure→Control Engines, or right-click an individual control engine on the Strategy Tree.

   The Configure Control Engines dialog box appears.



2. Click Add.

   The Select Control Engine dialog box appears.

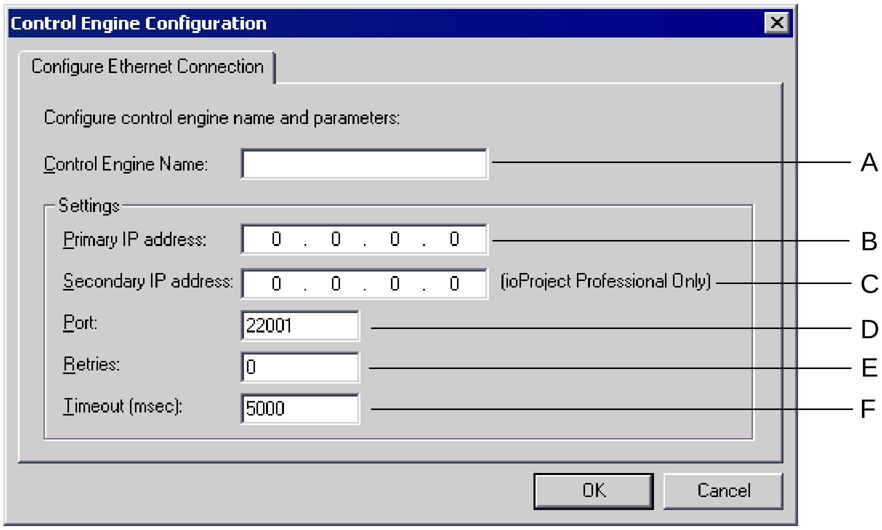


   This dialog box lists all the control engines configured on your system, whether or not they are associated with your strategy.

3. If the control engine you want appears in the list, it has already been defined on this PC, Click to highlight the control engine's name and click OK. Then skip to "Associating the Control Engine with Your Strategy" on page 5-4.

4. If the control engine you want is not in the list, click Add.

   The Control Engine Configuration dialog box appears.



Control Engine Configuration

Configure Ethernet Connection

Configure control engine name and parameters:

Control Engine Name: — A

Settings

Primary IP address: 0 . 0 . 0 . 0 — B

Secondary IP address: 0 . 0 . 0 . 0 (ioProject Professional Only) — C

Port: 22001 — D

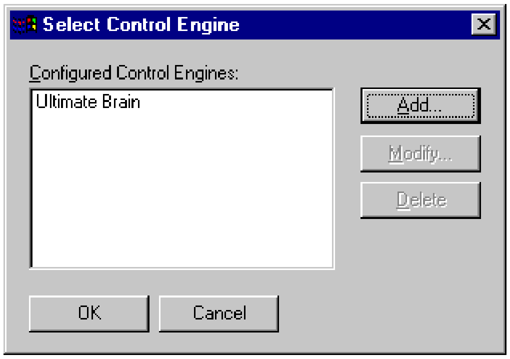
Retries: 0 — E

Timeout (msec): 5000 — F

5. Complete the fields as described in "Control Engine Configuration Dialog Box" below.

6. Click OK.

   In the Select Control Engines dialog box, the control engine appears in the list.



Select Control Engine

Configured Control Engines:

Ultimate Brain

7. In the Select Control Engine dialog box, highlight the control engine you want to associate with the strategy. Click OK. Continue with "Associating the Control Engine with Your Strategy" below.

Control Engine Configuration Dialog Box

(A) Control Engine Name Enter a descriptive name for the control engine. Valid characters are letters, numbers, spaces, and most other characters except colons and square brackets. Spaces cannot be used as first or last characters.

(B) IP Address Enter the IP address of the control engine in decimal notation (for example, `192.9.200.24`).

Pro (**C) Second IP** (ioControl Pro only) To configure Ethernet link redundancy, enter a secondary IP address. The secondary IP address can be the second Ethernet interface on a SNAP PAC controller, or a separate controller running a strategy designed to respond if the primary control engine is unavailable. Note that both IP addresses use the same port number. See "Configuring Ethernet Link Redundancy" on page 5-9 for more information.

(D) Port Enter the control engine's IP port number. The default of 22001 is the port of the host task on the control engine; this default normally should not be changed.

(E) Retries Retries indicate the number of times ioControl will reattempt communications with the control engine. Since retries are automatically handled by the protocol (TCP/IP), enter zero here.

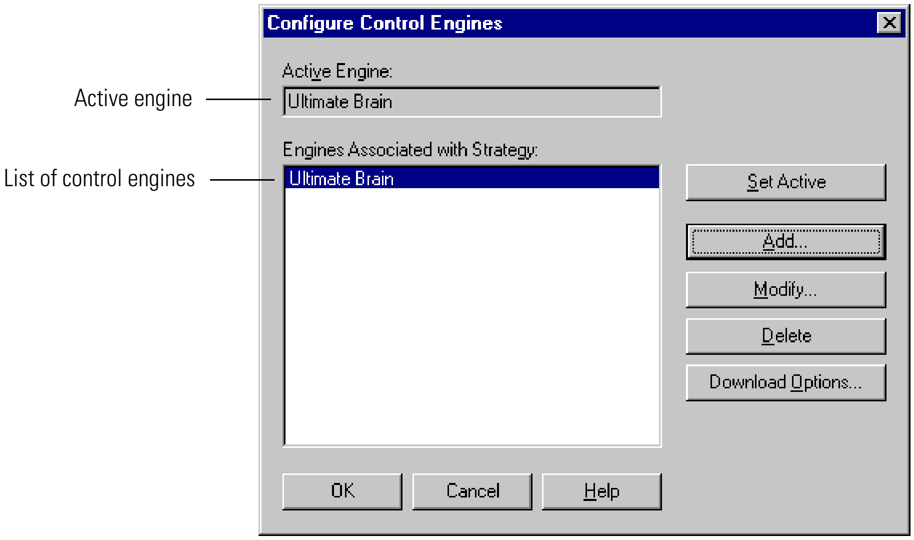
(F) Timeout Enter the timeout value in milliseconds. Timeout value is the length of time ioControl tries to establish communication through the port. If it fails, it tries again as many times as specified in D. Any positive integer is valid. For Ethernet, 3–5 seconds (3000–5000 milliseconds) is a good starting point.

Associating the Control Engine with Your Strategy

After you have defined the control engine on your PC (see page 5-2), it can be associated with your Strategy.

1. If you are in ioTerminal, close ioTerminal. Open the strategy in ioControl (Configure mode or Online mode) and choose Configure→Control Engine.

In the Configure Control Engines dialog box, the engine's name appears in the list.



2. Check to make sure the correct control engine appears in the Active Engine field. If not, highlight the one you want and click Set Active.

   Only one control engine can be active at any time. If only one control engine is listed, it automatically becomes the Active Engine.

Your control engine configuration is complete.

Using Network Segmenting in ioControl

You can take advantage of the two independent Ethernet network interfaces on a SNAP PAC controller to segment the control network from the company network. Because the two network interfaces are completely independent and have separate IP addresses, they can be set up on two separate networks. Host traffic can communicate on one, while I/O units are on the other.

To segment networks, start by assigning a secondary IP address in ioManager, following instructions in the *ioManager User's Guide*. The secondary IP address is used for communication through the controller's ENET2 interface. Remember that this interface must be on a completely separate network segment.

You don't need to do anything special in ioControl. Configure only one control engine, using the IP address of the Ethernet interface to be used for host communication. The control engine will direct communication to I/O units through the other interface, based upon the I/O unit's IP address.

NOTE: ioDisplay Basic and other hosts on the company network cannot communicate with I/O units when they are on a separate network. To solve this problem, you can purchase ioProject Professional, or ioDisplay Professional and OptoOPCServer (version 7.0 and newer) separately;

these applications support network segmenting by communicating with I/O units through the controller.

Using Ethernet Link Redundancy in ioControl

In ioControl Professional, you can configure a secondary control engine IP address for Ethernet link redundancy in case communication to the primary address fails. The secondary address can be the second Ethernet network interface on a SNAP PAC controller or a separate controller. If it is the second Ethernet network interface on a SNAP PAC controller, you must assign the secondary IP address in ioManager in order to communicate with it. See the *ioManager User's Guide* for instructions.

NOTE: If you use a separate controller for link redundancy, be aware that coordinating strategy data on two controllers can be difficult, and no automatic method for doing so is available. Using a separate controller for monitoring over a redundant link is usually less difficult to coordinate than using a separate controller for control over a redundant link.

The two Ethernet interfaces on a SNAP PAC controller are ideal for adding redundant Ethernet network links to your control system. An ioDisplay Professional project or OptoOPCServer, for example, can be set up to use the primary and secondary addresses to access data from the ioControl strategy running on the control engine. If the primary address is unavailable, the client application will automatically shift to the secondary address; if the secondary address then fails, it will automatically try the primary address again. (You can also manually change the address, from within the client application. See the OptoOPCServer or ioDisplay user's guide for details.)

ioControl Professional also provides for Ethernet link redundancy from the controller to Ethernet-based I/O units. You can configure a secondary I/O unit IP address in case communication with the primary address fails. The secondary address can be the second Ethernet network interface on a SNAP PAC R-series controller, or a separate I/O unit. (If you use a separate I/O unit, you may find that wiring is problematical.) See "Commands for Ethernet Link Redundancy" on page 10-15 for more information on primary and secondary I/O units.

System Architecture for Ethernet Link Redundancy

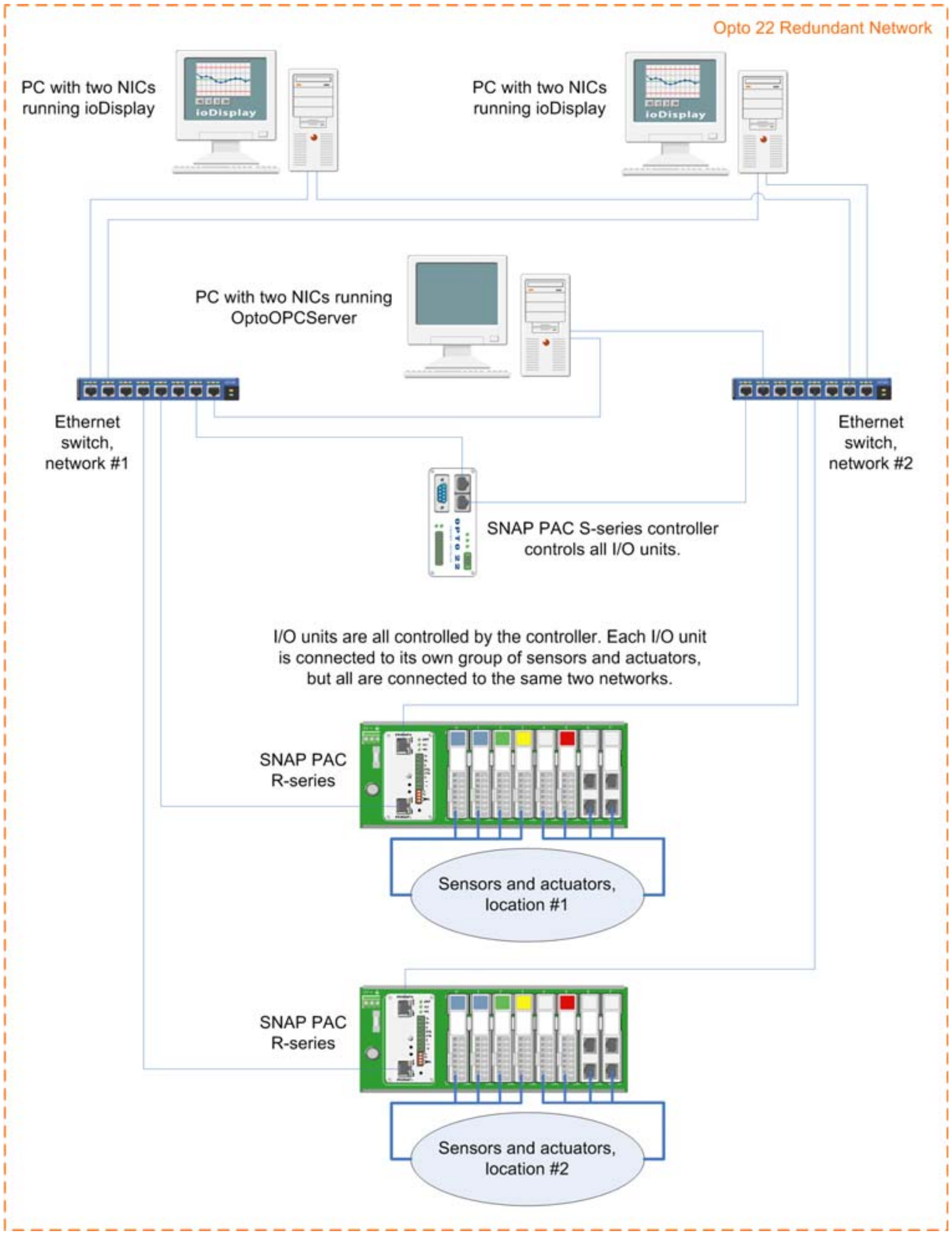
You can set up your system in several ways to take advantage of the link redundancy capability in ioControl. The following diagrams show some of those ways. For additional information, see the user's guides for the SNAP PAC controllers. For details on how ioDisplay and OptoOPCServer work with link redundancy, see their user's guides.

Ethernet Link Redundancy

In this example, the primary concern is that the Ethernet network may need maintenance or may fail, leaving the computer running OptoOPCServer, the PC running ioDisplay Professional, the controller, and the I/O units unable to communicate.

This solution is to connect all these devices on two networks; if one network goes down, devices can communicate on the other. Note that each computer has two network interface cards (NICs).

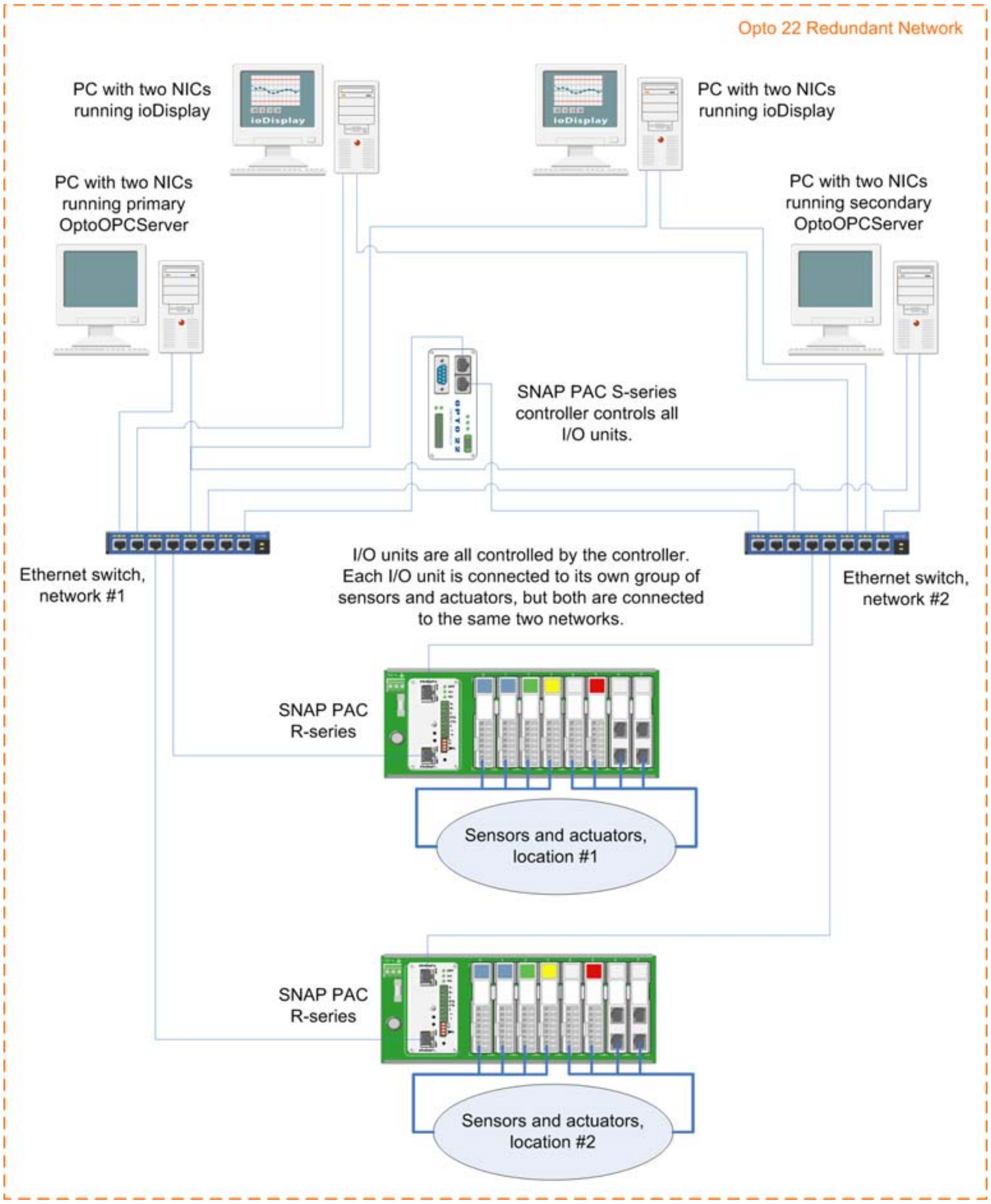
If one network needs maintenance, for example to replace a switch, you can safely shut it down using the Set Target Address commands in ioControl and options in ioDisplay Professional and OptoOPCServer.


Opto 22 Redundant Network
PC with two NICs running ioDisplay
ioDisplay
PC with two NICs running ioDisplay
ioDisplay
PC with two NICs running OptoOPCServer
Ethernet switch, network #1
Ethernet switch, network #2
SNAP PAC S-series controller controls all I/O units.
I/O units are all controlled by the controller. Each I/O unit is connected to its own group of sensors and actuators, but all are connected to the same two networks.
SNAP PAC R-series
Sensors and actuators, location #1
SNAP PAC R-series
Sensors and actuators, location #2

Ethernet Link, Computer, and Software Redundancy

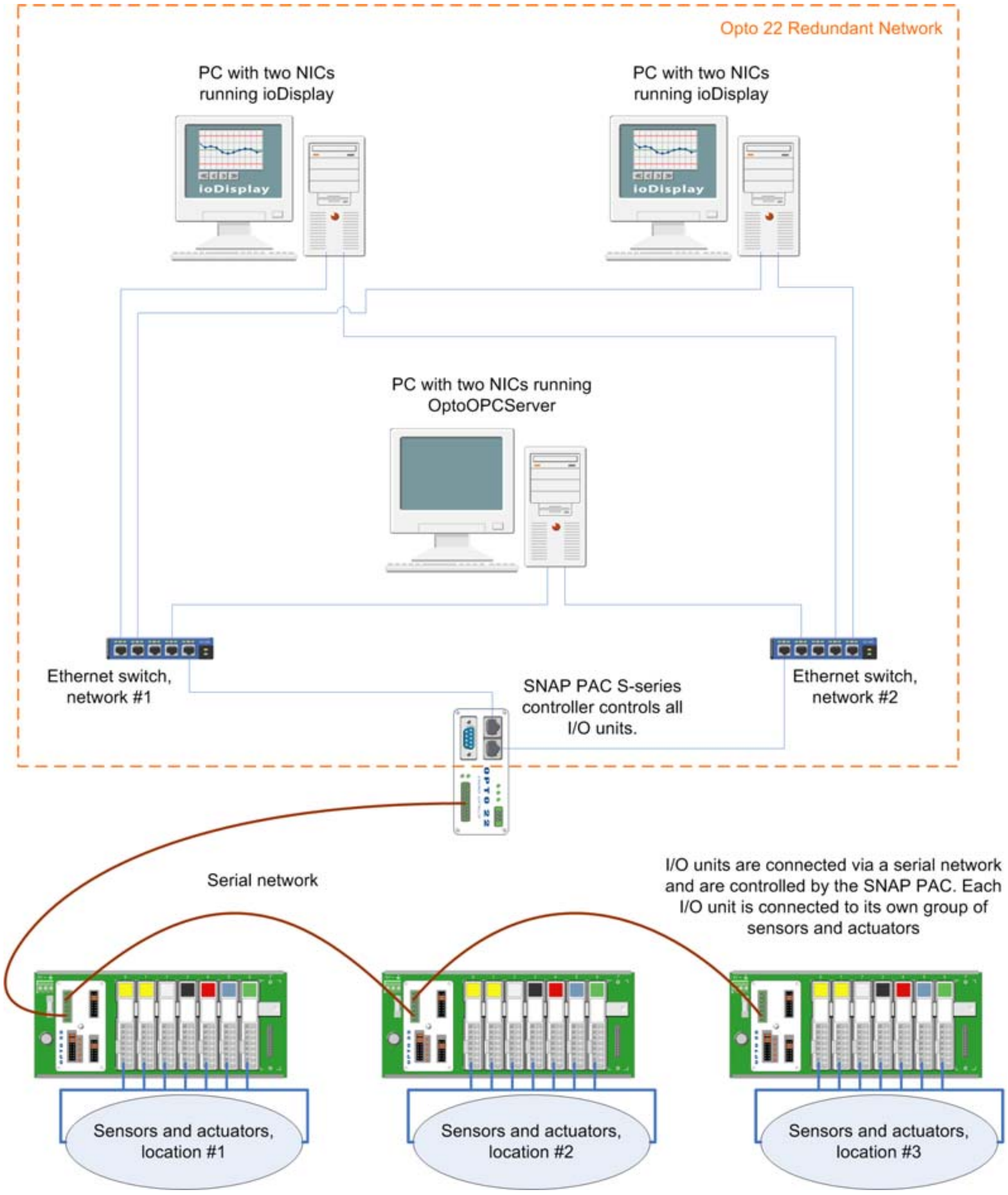
The second example of link redundancy illustrates concern not only about the stability of the Ethernet network, but also about the computers and the software run on them. Any of these things—the network, a computer, or the OptoOPCServer or ioDisplay Professional software running on the computer—may need maintenance or may fail.

The solution here is to provide duplicate computers and software, and connect all devices on two networks. Each computer has two network interface cards (NICs).


Opto 22 Redundant Network
PC with two NICs running ioDisplay
PC with two NICs running ioDisplay
PC with two NICs running primary OptoOPCServer
PC with two NICs running secondary OptoOPCServer
SNAP PAC S-series controller controls all I/O units.
Ethernet switch, network #1
I/O units are all controlled by the controller. Each I/O unit is connected to its own group of sensors and actuators, but both are connected to the same two networks.
Ethernet switch, network #2
SNAP PAC R-series
Sensors and actuators, location #1
SNAP PAC R-series
Sensors and actuators, location #2

Ethernet Link Redundancy with Serial I/O Units

This third example of link redundancy shows a SNAP PAC S-series controller with serial I/O units. The redundancy is in the Ethernet networks between the controller and computers running ioDisplay Professional and OptoOPCServer. This example provides link redundancy for the Ethernet network and a separate serial control network. Again, note that the computers all have two NICs.


Opto 22 Redundant Network
PC with two NICs running ioDisplay
PC with two NICs running ioDisplay
ioDisplay
ioDisplay
PC with two NICs running OptoOPCServer
Ethernet switch, network #1
Ethernet switch, network #2
SNAP PAC S-series controller controls all I/O units.
Serial network
I/O units are connected via a serial network and are controlled by the SNAP PAC. Each I/O unit is connected to its own group of sensors and actuators
Sensors and actuators, location #1
Sensors and actuators, location #2
Sensors and actuators, location #3

Pro

Configuring Ethernet Link Redundancy

Ethernet link redundancy to the control engine is easy to configure. First, make sure the secondary IP address has been assigned in ioManager. Then in ioControl, just enter both the primary and secondary IP addresses when you configure the control engine (see page 5-1). It is not necessary to separately define the second controller on your PC.

To configure link redundancy from the control engine to I/O units, make sure the secondary IP address has been assigned in ioManager. Then enter both the primary and secondary IP addresses when you configure the I/O unit (see “Adding an I/O Unit” on page 6-12).

Pro

Using Strategies with Link Redundancy

Although clients of ioControl strategies such as ioDisplay and OptoOPCServer automatically shift to the secondary address if the primary address is not available, the ioControl debugger does not. In ioControl, you control which address receives communication, so you know exactly how communication is occurring during debugging. To change between primary and secondary control engine addresses, you must be in Configure or Online mode. Follow the steps in “Changing the Control Engine that Receives the Downloaded Strategy” on page 5-11.

Once you have changed the address, when you enter Debug mode, you are communicating through the address you chose. Downloading the strategy to a SNAP PAC controller through one of its Ethernet interfaces replaces any strategy that was downloaded earlier through its other interface.

Changing or Deleting a Control Engine

See the following topics to change or delete a control engine:

- “Changing a Control Engine’s Definition” (below)
- “Changing the Control Engine that Receives the Downloaded Strategy” on page 5-11
- “Removing a Control Engine’s Association with a Strategy” on page 5-11
- “Deleting a Control Engine from Your PC” on page 5-12

Changing a Control Engine’s Definition

Whenever necessary, you can change a control engine’s definition on your PC—its name, the port the PC uses to communicate with it, or the PC port setup (such as timeouts and retries). These changes can be made either in ioControl or in the ioTerminal utility.

1. Choose one of the following:
   - **In ioControl:** With the Strategy open in Configure mode or Online mode, right-click the control engine name on the Strategy Tree and choose Modify from the pop-up menu.
   - **In ioTerminal:** Choose Start→Programs→Opto 22→ioProject Software→Tools→ioTerminal. From the Configure menu, choose Control Engine.

Select Control Engine dialog box appears.

2. In the Select Control Engine dialog box, click the control engine you want to change and click Modify.

3. Make the necessary changes, following the same steps you would for configuring the control engine initially. (For help, see "Defining a Control Engine on Your PC" on page 5-2.)

Changing the Control Engine that Receives the Downloaded Strategy

You can configure several control engines for a Strategy, but only one at a time can receive the downloaded strategy. The control engine that receives the strategy is called the active engine.

NOTE: In ioControl Professional, you can also configure a control engine for Ethernet link redundancy (see "Using Ethernet Link Redundancy in ioControl" on page 5-6 for more information). In this case, you can also choose whether the primary or secondary IP address will receive the strategy.

To change the control engine that receives the downloaded strategy, follow these steps:

1. Make sure the strategy is open and in Configure or Online mode.

2. On the Strategy Tree, right-click the name of the control engine you want to set as the active engine.

   If its name does not appear in the Strategy Tree, follow the steps in "Configuring Control Engines" on page 5-1.

3. From the pop-up menu, choose Set Active.

   The active engine moves to the top of the list in the Strategy Tree.

4. I(ioControl Pro only) If the control engine is configured for Ethernet link redundancy, right-click the name of the control engine on the Strategy Tree again. From the pop-up menu, choose Use Primary IP Address or Use Secondary IP Address.

Removing a Control Engine's Association with a Strategy

If you no longer want to use a control engine with a Strategy, you can remove its association with the strategy. This action does not delete the control engine's definition on your PC.

***CAUTION:** Do not delete the control engine from within the Select Control Engine dialog box. Doing so will delete it from the PC as well as the strategy.*

1. Make sure the strategy is open in Configure mode or Online mode.

2. On the Strategy Tree, right-click the name of the control engine you want to remove. From the pop-up menu, choose Delete.

   The control engine is no longer associated with your strategy.

Deleting a Control Engine from Your PC

If you are sure that a control engine will no longer be used with your PC, you can delete it using the ioTerminal utility. Deleting the control engine removes its definition only on the PC you are using.

1. Choose Start→Programs→Opto 22→ioProject Software→Tools→ioTerminal.
2. Right-click the name of the control engine in the list.

   CAUTION: *Make sure you are highlighting the right one. You cannot undo a deletion.*

3. From the pop-up menu, choose Delete.

   The control engine is no longer defined on the PC.

Inspecting Control Engines and the Queue

You may want to inspect or change control engines while you are running the Strategy in Debug mode. See the following topics to view control engine information and the engine's message queue, either from ioControl in Debug mode or from the ioTerminal utility:

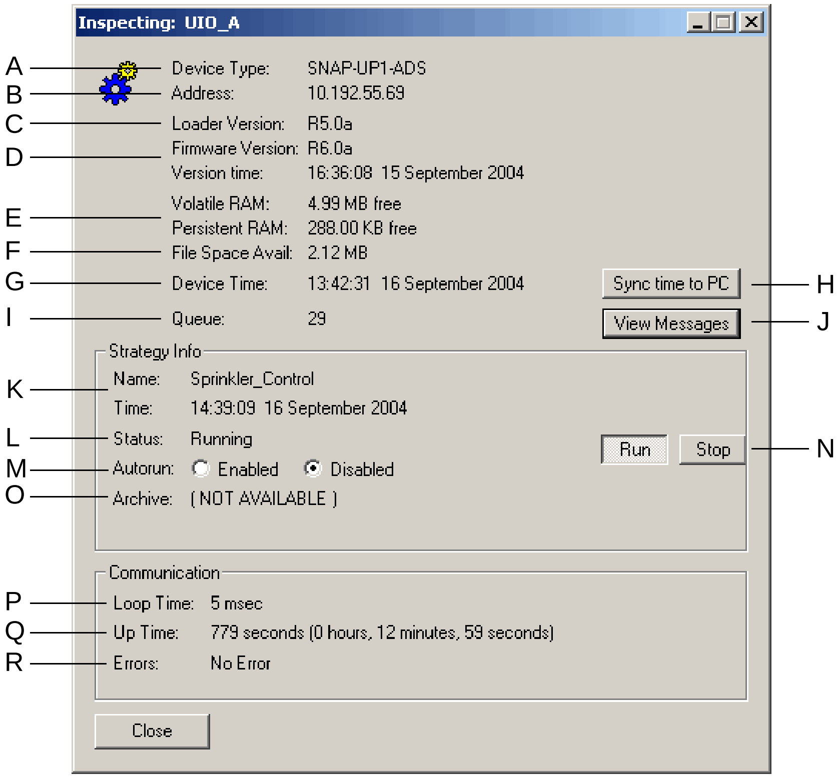
- "Inspecting Control Engines in Debug Mode" (below)
- "Viewing the Message Queue" on page 5-14
- "Inspecting Control Engines from the ioTerminal Utility" on page 5-16

Inspecting Control Engines in Debug Mode

With the Strategy running in Debug mode, click the Inspect Control Engine button in the toolbar.

You can also double-click the active engine (the first one under the Control Engines folder) on the Strategy Tree, or right-click the control engine and choose Inspect from the pop-up menu, or select Control Engine→Inspect. Also, if a blue INFO, yellow WARNING, or red ERROR box appears in the strategy's status bar, you can click the box.

The Inspecting dialog box opens.



Inspecting: UIO_A

Device Type: SNAP-UP1-ADS
Address: 10.192.55.69
Loader Version: R5.0a
Firmware Version: R6.0a
Version time: 16:36:08 15 September 2004
Volatile RAM: 4.99 MB free
Persistent RAM: 288.00 KB free
File Space Avail: 2.12 MB
Device Time: 13:42:31 16 September 2004
Queue: 29

Strategy Info
Name: Sprinkler_Control
Time: 14:39:09 16 September 2004
Status: Running
Autorun: ○ Enabled ● Disabled
Archive: (NOT AVAILABLE)

Communication
Loop Time: 5 msec
Up Time: 779 seconds (0 hours, 12 minutes, 59 seconds)
Errors: No Error

Here you see data relating to the control engine. If you are using Ethernet link redundancy in ioControl Professional, remember that the information shown is for the IP address you chose before entering Debug mode. (See "Using Strategies with Link Redundancy" on page 5-10 for more information.)

(A) Device Type Type of device the control engine is running on

(B) Address Device's IP address

(C) Loader Version Version of the loader software. The loader is used to download firmware to the device.

(D) Firmware Version Version number of the firmware (kernel) loaded on the device, and the date the firmware was released

(E) Volative and Persistent RAM Amount of memory (RAM) available on the control engine. For example, a SNAP Ultimate brain has a total of 8 MB of RAM on the control side, 256 KB of which is battery-backed. Volatile RAM shows the amount of total RAM available for use. Persistent RAM shows the amount available in battery-backed RAM, where persistent variables, variables initialized on download, the autorun flag, and the strategy archive are stored.

(F) File Space Avail Space available in the control engine's file system (See the *ioManager User's Guide* for more information about the file system.)

(G) **Device Time** Current date and time recorded on the control engine

(H) **Sync time to PC** Click to synchronize the control engine's time and date to that of the PC running ioControl, click the Sync to PC's Time/Date button.

(I) **Queue** Number of messages (information, warning, and error messages) encountered when attempting to run the strategy on the control engine (up to a maximum of 1000 messages). (See "" on page 5-14.)

(J) **View Messages** Click to open the View Messages dialog box, listing the details of any information, warning, or error messages.

(K) **Name and Time** Name of the strategy currently running on the control engine, and the date and time it was downloaded

(L) **Status** Current status of the strategy

(M) **Autorun** Click to indicate whether the strategy should automatically run when the control engine is restarted. The strategy must be stored to flash to autorun. See page 7-4 for more information.

(N) **Run/Stop** Buttons to start or stop the strategy. This example shows the strategy running.

(O) **Archive** Information about the strategy currently archived on the control engine

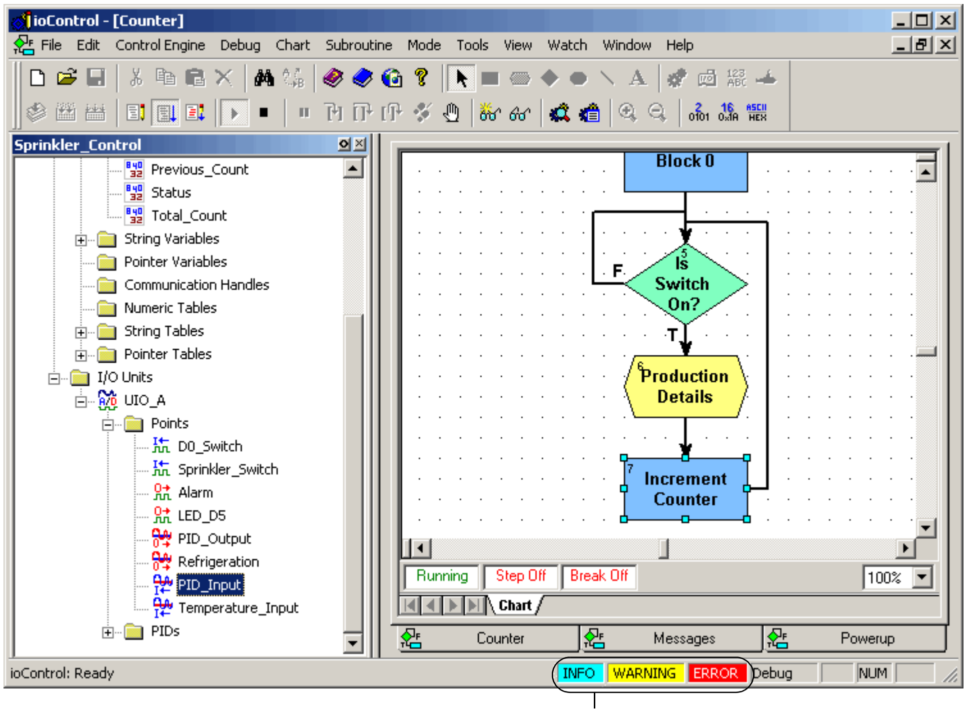
(P) **Loop Time** Time required to gather the inspection data from the control engine (the time taken for a single transaction)

(Q) **Up Time** Total time that the control engine has been running since powerup

(R) **Errors** Any communication errors

Viewing the Message Queue

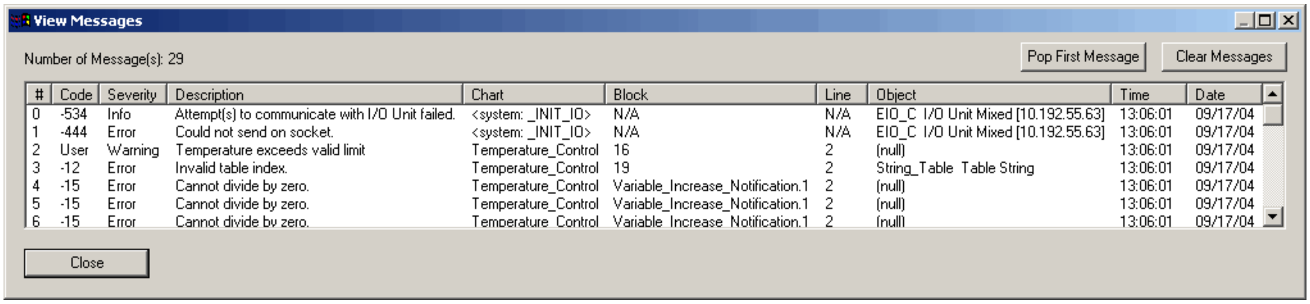
The message queue holds error, information, and warning messages. All may be helpful in troubleshooting. When a message is placed in the queue, a blue INFO, yellow WARNING, or red ERROR box appears in the ioControl status bar, as shown in the following diagram.



Messages have been placed in the queue.

To see the message queue,

1. Click the INFO, WARNING, or ERROR box, or click the Inspect Control Engine button in the toolbar.
2. In the Inspect Control Engine dialog box, click the View Messages button.



Number of Message(s): 29

#	Code	Severity	Description	Chart	Block	Line	Object	Time	Date
0	-534	Info	Attempt(s) to communicate with I/O Unit failed.	<system: _INIT_IO>	N/A	N/A	EIO_C I/O Unit Mixed [10.192.55.63]	13:06:01	09/17/04
1	-444	Error	Could not send on socket.	<system: _INIT_IO>	N/A	N/A	EIO_C I/O Unit Mixed [10.192.55.63]	13:06:01	09/17/04
2	User	Warning	Temperature exceeds valid limit	Temperature_Control	16	2	(null)	13:06:01	09/17/04
3	-12	Error	Invalid table index.	Temperature_Control	19	2	String_Table Table String	13:06:01	09/17/04
4	-15	Error	Cannot divide by zero.	Temperature_Control	Variable_Increase_Notification.1	2	(null)	13:06:01	09/17/04
5	-15	Error	Cannot divide by zero.	Temperature_Control	Variable_Increase_Notification.1	2	(null)	13:06:01	09/17/04
6	-15	Error	Cannot divide by zero.	Temperature_Control	Variable_Increase_Notification.1	2	(null)	13:06:01	09/17/04

Drag the edge of a column heading to see all the information in the column. See Message Queue Information below.

3. To delete the top (oldest) message on the list, click Pop First Message.
4. To delete all messages, click Clear Messages.
5. Close the dialog box to return to the Inspect Control Engine dialog box.

   Any changes you have made to the queue are reflected there.

Message Queue Information

Each message in the View Messages dialog box includes the following information:

Code The message or error code number (see "List of Common Messages" on page B-3) or *User* if the message was placed in the queue using the command Add Message to Queue.

Severity Information, Warning, or Error.

Chart and Block The chart and block being executed when the error occurred. If the error occurred someplace outside the strategy, for example when trying to connect to an I/O unit, Chart shows <system>. If the error occurred in a subroutine, Chart shows the chart that called the subroutine, and Block indicates the name of the subroutine plus the block number in the format `<sub name>.<block number>`. For example, error #4 above ("Cannot divide by zero") occurred in block 1 of the subroutine Variable_Increase_Notification, which was called by the Temperature_Control chart.

Line If you are in Full Debug mode, the line being executed when the error occurred.

Object The table, I/O unit, or other object affected by the message. In errors #1 and #2 above, the control engine was unable to communicate with I/O unit EIO_C. The unit's IP address is shown for easy reference.

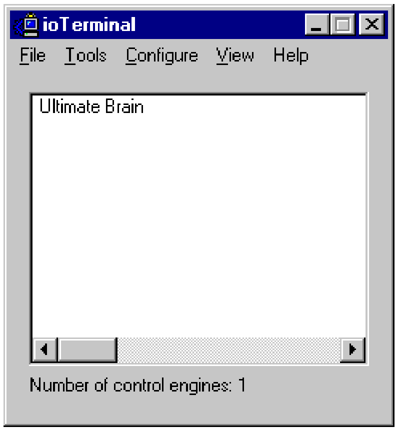
Time and Date When the error occurred.

Inspecting Control Engines from the ioTerminal Utility

You can also inspect control engines from the ioTerminal utility.

1. Click the Windows Start menu and select Programs→Opto22→ioProject Software→Tools→ioTerminal.

   The ioTerminal window appears.



Ultimate Brain

Number of control engines: 1

2. Double-click the control engine you want to see (or right-click it and choose Status from the pop-up menu).

   The Inspect Control Engine dialog box appears. The dialog box is explained on page 5-12.

Downloading Files to the Control Engine

This section discusses how to archive strategies and download Forth files directly related to a Strategy, such as library or initialization files. For information on using the brain's file system to store data and manipulate it within your strategy, see page 10-42.

Archiving Strategies

Archiving strategies on the control engine provides a backup in case original Strategy files on the computer are lost. Archive files are date and time stamped, and zipped for compact storage. The archive file name on the control engine is in one of the following formats:

Path\Filename.Download.D02282000.T114351.zip
Path\Filename.Online.D02282000.T114351.zip

The date stamp (D) is in the format mm/dd/yyyy. In the examples above, the date is February 28, 2000. The time stamp (T) is in the format hh/mm/ss. In the examples above, the time is 51 seconds past 11:43 A.M.

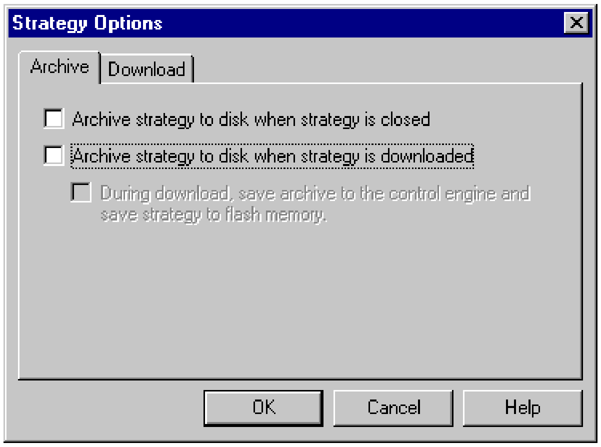
Archiving to the Control Engine

When you archive a Strategy to the control engine, you are placing the zipped file in battery-backed RAM. If power to the control engine is lost, the archive is still there. Archiving to the control engine as well as the computer makes sure that an older strategy can always be found and updated, even after personnel changes occur and years pass.

Make sure the control engine has sufficient memory available to store the archive file. Since only one strategy can be on the control engine at any time, only the latest archive for that strategy is on the control engine. Other archives are erased during strategy download.

Follow these instructions to archive a strategy to the control engine:

1. In ioControl, choose File➞Strategy Options.

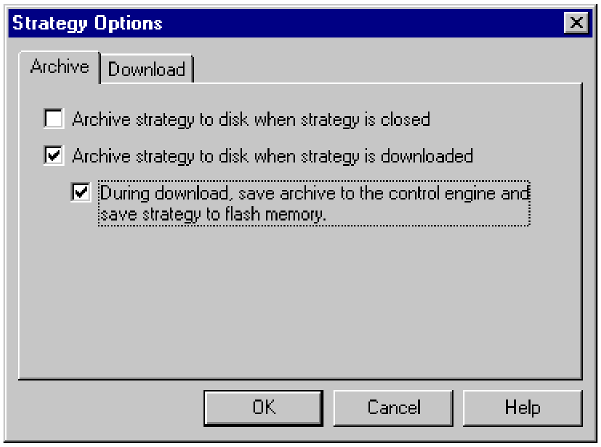


☐ Archive strategy to disk when strategy is closed

☐ Archive strategy to disk when strategy is downloaded

☐ During download, save archive to the control engine and save strategy to flash memory.

2. In the Strategy Options dialog box, make sure the Archive tab is on top. Click Archive strategy to disk when strategy is downloaded. Also click “During download, save archive to the control engine and save strategy to flash memory.”



☐ Archive strategy to disk when strategy is closed

☑ Archive strategy to disk when strategy is downloaded

☑ During download, save archive to the control engine and save strategy to flash memory.

3. Click OK.

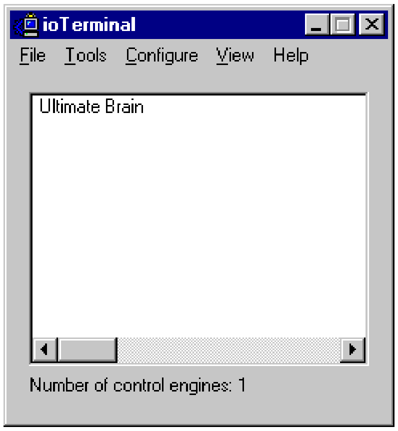
   The strategy will be archived to the computer and to the control engine when it is downloaded. Any archive already on the control engine will be replaced by the new archive. In addition, the strategy will be saved to flash memory, so it will still be available if power to the controller or SNAP Ultimate brain is turned off.

Restoring Archived Strategies from the Control Engine

If original Strategy files are lost or damaged, you can use ioTerminal to restore the strategy archive from the control engine to a computer.

1. Click the Windows Start menu and choose Programs➞Opto22➞ioProject Software➞Tools➞ioTerminal.

The ioTerminal window appears.



2. Right-click the control engine and choose Upload→Strategy Archive from the pop-up menu.

3. In the Save ioControl Strategy Archive As dialog box, navigate to the folder where you want to save the archive file. Keep the file name as it is, so you can see the date and time it was originally downloaded to the control engine. Click Save.

   A dialog box shows progress as the archive file is uploaded to the computer.

4. Navigate to the zipped archive file. Assuming you are using WinZip, double-click the file name. Highlight all files and click Extract. Extract them to the location you want.

5. When all files are extracted, double-click the .idb file to open the strategy. If necessary, re-link subroutines and files run before or after a strategy.

   Re-linking may be necessary because the directory structure in the zip file may not match what was originally on the computer. The zip file structure is as follows:

   Root (.idb, chart files, .inf)
   - Subroutines
   - Control engine files
     - Control_Engine_Name_1
       - Before run file
       - After run file
     - Control_Engine_Name_2
       - Etc.

Downloading Files Without Opening ioControl

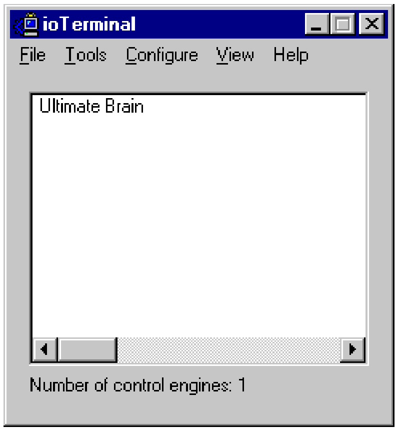
Using the ioTerminal utility, you can download ioControl strategies or Forth files related to the Strategy, such as library or initialization files, directly to a control engine, without having to open each program.

(For information on using the brain's file system to store data and manipulate it within your strategy, see page 10-42.)

NOTE: If you are downloading an ioControl strategy that requires other files, be sure to download the files in the correct order (for example, library file, then strategy file, then initialization file). If you need to set initial values for individual table elements on strategy download only, see page 9-10.

1. Click the Windows Start menu and choose Programs➞Opto22➞ioProject Software➞Tools➞ioTerminal.

   The ioTerminal window appears.



2. Right-click the control engine and choose Download➞Forth Files from the pop-up menu.
3. In the Download File dialog box, click Browse.
4. In the Open dialog box, locate the file you want to download. When the full path appears, click OK.

   If necessary to find the file, choose All Files from the Files of Type drop-down menu.

   The download begins, and a dialog box shows its progress.

CHAPTER 6

Working with I/O

Introduction

In addition to configuring a control engine to run your strategy, you also need to configure input/output hardware to do the work: turning things on, setting temperatures, monitoring controls, and so on.

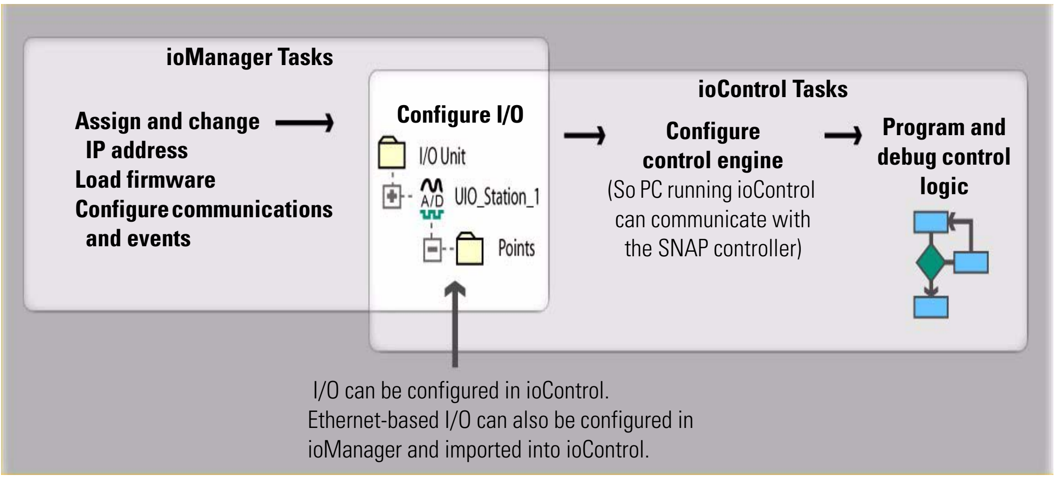
This chapter shows you how to configure and work with I/O units, I/O points, and PID loops.

In this Chapter

Choosing a Configuration Tool

Configuring I/O is one of your major planning steps in developing an ioControl Strategy. Generally it's best to configure all I/O units, points, and PID loops at once, before you start building flowcharts.

There are two tools you can use for configuration: ioControl and ioManager. These two tools serve different purposes, but they overlap when it comes to configuring I/O. The graphic on the next page compares their functions.


ioManager Tasks
Assign and change IP address
Load firmware
Configure communications and events
Configure I/O
I/O Unit
UIO_Station_1
A/D
Points
ioControl Tasks
Configure control engine
(So PC running ioControl can communicate with the SNAP controller)
Program and debug control logic

I/O can be configured in ioControl. Ethernet-based I/O can also be configured in ioManager and imported into ioControl.

I/O units and points must be configured to match the ioControl strategy you will run. You can configure most Ethernet-based I/O unit and point functions either in ioControl or in ioManager.

IMPORTANT: *For E1 and E2 I/O units, you must use ioManager. See form #1576, Technical Note: I/O Configuration for E1 and E2 Brain Boards, for instructions.*

For ***mistic I/O units****, ioManager cannot be used for configuration. Use ioControl.*

For most I/O units, if you are already in ioControl, configuration is easier there and you can use the loopback IP address for SNAP Ultimate I/O units controlling themselves. However, some functions for Ethernet-based I/O units cannot be configured in ioControl.

If you use ioManager, you can save your configuration to a file, load it to multiple I/O units at once, and use it for referencing points in OPC. However, you cannot use the loopback address in ioManager and you cannot use ioManager for mistic I/O units.

Choose your configuration tool based on what you need to do:

Use ioControl for I/O configuration if	Use ioManager for I/O configuration if
• You have only one I/O unit or I/O unit configurations are different. • You are configuring mistic I/O units. • The strategy will run on SNAP PAC R-series or SNAP Ultimate I/O units that are controlling themselves using the loopback IP address, 127.0.0.1 • You are using an Ethernet network for communications. (Exception: Use ioManager for E1 or E2 I/O units.) • The strategy handles all logic; you are not also configuring events and reactions on I/O units.	• You have multiple I/O units whose configurations are exactly the same or similar. • You have an E1 or E2 I/O unit. • You are using a modem connection (PPP) or SNMP. • You are using event messages or email. • You are configuring events and reactions on the I/O unit in addition to strategy logic. • You are using OPC to communicate with I/O units. • You are not using ioControl.

Whichever tool you use for configuring I/O, be aware of the impact if you later change configuration. For example, if you configure I/O in ioManager, download the configuration file to I/O units, and then later add a point in ioControl, remember that your configuration file doesn’t contain that point.

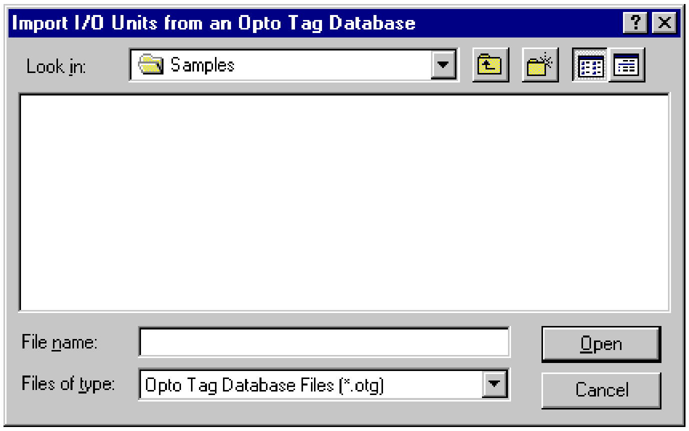
If you use ioManager, follow instructions in Opto 22 form #1440, the *ioManager User's Guide*. When you have finished configuration and saved the configuration file, you can import it into ioControl following the steps in "Importing I/O Configuration into ioControl" below.

If you use ioControl, follow the steps beginning with "About I/O Units" on page 6-4.

Importing I/O Configuration into ioControl

If you have configured all I/O units, points, and PID loops in ioManager, follow these steps to import the configuration file into an ioControl Strategy.

1. Open the strategy in ioControl. In the Strategy Tree, right-click the I/O units folder. From the pop-up menu, choose Import.


Import I/O Units from an Opto Tag Database
?
Look in:
Samples
File name:
Files of type:
Opto Tag Database Files (*.otg)
Open
Cancel

2. Navigate to the configuration file you created and saved in ioManager. Double-click it to open it.

   The configuration information is imported. You can expand the I/O units folder to see the imported units and their points.

   If you need to configure additional I/O units from within ioControl, see "About I/O Units" on page 6-4. To tune PID loops, see "Inspecting and Tuning PID Loops" on page 6-58.

Copying I/O Configurations

If you have two strategies that use similar I/O units and points, you can export an I/O configuration from one Strategy into a file, and then import it into the other strategy.

If you need similar configurations for several Ethernet-based I/O units, you can use ioManager to send it to multiple I/O units at once. (You cannot use ioManager for serial-based I/O units.) For more information on using ioManager, see Opto 22 form #1440, the *ioManager User's Guide*.

Creating the Configuration Export File

1. Open the strategy you are copying from in ioControl. In the Strategy Tree, right-click the I/O Units folder and choose Export from the pop-up menu.

   The Export I/O Units to an Opto Tag Database dialog box appears.

2. Navigate to the location where you want to place the export file. Type the file name, and click Save.

   The export file is created. It is a comma-delimited ASCII file. If you wish, you can open it in Notepad or Excel.

Importing the Configuration File

When you import the I/O configuration file, it does not delete any I/O units or points that are already there. If the import file contains I/O units with the same names as those already in the Strategy, you can choose whether to update them. Updating changes points that have the same name and adds new points, but does not delete points.

1. Open the strategy into which you want to import the I/O configuration.

2. In the Strategy Tree, right-click I/O Units and choose Import from the pop-up menu.

3. Navigate to the location of the export file you created. Highlight its name and click Open.

   The I/O units and points are updated from the configuration file. To see them, click the plus sign next to the I/O Units folder on the Strategy Tree.

About I/O Units

In ioControl, the term *I/O unit* usually refers to a mounting rack with a brain or brain board and up to 16 I/O modules attached. The following table shows brains, racks, and I/O modules that can be used in ioControl:

Brain Part Number	Compatible Racks	Max # Modules	Module types
The following I/O units are supported in ioControl Basic and ioControl Professional:			
SNAP-UP1-ADS SNAP-B3000-ENET SNAP-ENET-RTC	SNAP-B4M	4	SNAP analog, digital, and special-purpose
	SNAP-B8M SNAP-B8MC SNAP-B8MC-P	8	
	SNAP-B12M SNAP-B12MC SNAP-B12MC-P	12	SNAP analog, digital, and special-purpose (standard digital in positions 0–7 only; high-density digital in any position)
	SNAP-B16M SNAP-B16MC SNAP-B16MC-P	16	

Brain Part Number	Compatible Racks	Max # Modules	Module types
SNAP-UP1-D64 SNAP-ENET-D64	SNAP-D64RS	16	SNAP digital (limited digital functions; no high-density digital)
SNAP-UP1-M64 SNAP-ENET-S64	SNAP-M16	4	SNAP analog, digital, and special-purpose (limited digital functions; standard and high-density digital modules OK in any position)
	SNAP-M32	8	
	SNAP-M64	16	
The following I/O units are supported in ioControl Professional Only:			
B3000 (serial)	SNAP-B4M	4	SNAP analog and standard digital (no high density digital; no serial)
	SNAP-B8M SNAP-B8MC SNAP-B8MC-P	8	
	SNAP-B12M SNAP-B12MC SNAP-B12MC-P	12	SNAP analog and standard digital (no high density digital; no serial; standard digital in positions 0–7 only)
	SNAP-B16M SNAP-B16MC SNAP-B16MC-P	16	
SNAP-BRS	SNAP-B8M SNAP-B8MC SNAP-B8MC-P	8	SNAP standard digital only (no high-density digital. Max. 32 points)
G4D16R	Brick	16	G4 digital (max. 16 points)
G4D32RS	Brick	32	G4 digital (max. 32 points)
G4A8R	Brick	8	G4 analog (max 8 points
B100	PB16 G4PB16	16	G1, G4, or Quad Pak digital (max. 16 points)
B200	PB16A	16	G1 analog (max 16 points)

For most serial-based brain boards, racks are limited to 16 points of either analog or digital I/O, so each I/O unit is either digital or analog. Racks for Ethernet-based brains, however, hold up to 64 points and can be either digital only or both analog and digital. If the rack attached to a SNAP Ethernet-based brain accommodates both analog and digital modules, the I/O unit includes both analog and digital modules.

In most cases the "I/O unit" you configure in ioControl is the same as the physical I/O unit (rack, brain, and I/O modules). The entire rack of points is configured as one I/O unit, because that's how the points are addressed by the brain.

(ioControl) In two cases, however, the "I/O unit" in ioControl does not exactly correspond to the physical I/O unit, because these brains address their I/O modules in a different way:

- **A SNAP B3000 serial brain** addresses up to four groups of 16 points on the largest rack. Each group of 16 points must be configured as a separate I/O unit, either analog or digital.

(Some ioControl commands communicate with all the points on one I/O unit at once. For more information on these commands, see "Using I/O Unit Commands" on page 4-23.)

- **A G4D32RS brick** contains the equivalent of two 16-module units, and the brain board addresses them separately. When you configure it in ioControl, notice that it is called a G4D16RS. Configure two G4D16RS I/O units for each brick.

Addressing I/O Units

To configure I/O, you must know how the brain addresses its I/O points. The following pages show module and point numbers for the brains listed below.

IMPORTANT: *These diagrams show addressing for analog and standard SNAP digital modules only. High-density digital modules do not require configuration. Serial modules are not configured unless you need to change communication parameters; if so, use ioManager.*

For this brain	See
SNAP-PAC-R1 SNAP-UP1-ADS SNAP-B3000-ENET SNAP-ENET-RTC	page 6-6
SNAP-PAC-R2 SNAP-UP1-M64 SNAP-ENET-S64	page 6-8
SNAP-UP1-D64 SNAP-ENT-D64	page 6-9
B3000 (serial) SNAP-BRS	page 6-9
G4 bricks B100 B200	page 6-12
E1 E2	See form #1576, *I/O Configuration for E1 and E2 Brain Boards*

SNAP Ethernet Analog and Digital Systems

Processor	Compatible Racks
SNAP-PAC-R1	SNAP M-series (SNAP-M16, SNAP-M32, SNAP-M48, SNAP-M64)
SNAP-UP1-ADS SNAP-B3000-ENET SNAP-ENET-RTC	SNAP-B series (SNAP-B4, SNAP-B8, SNAP-B12; SNAP-B16, SNAP-B8MC, SNAP-B12MC, SNAP-B16MC, SNAP-B8MC-P, SNAP-B12MC-P, SNAP-B16MC-P)

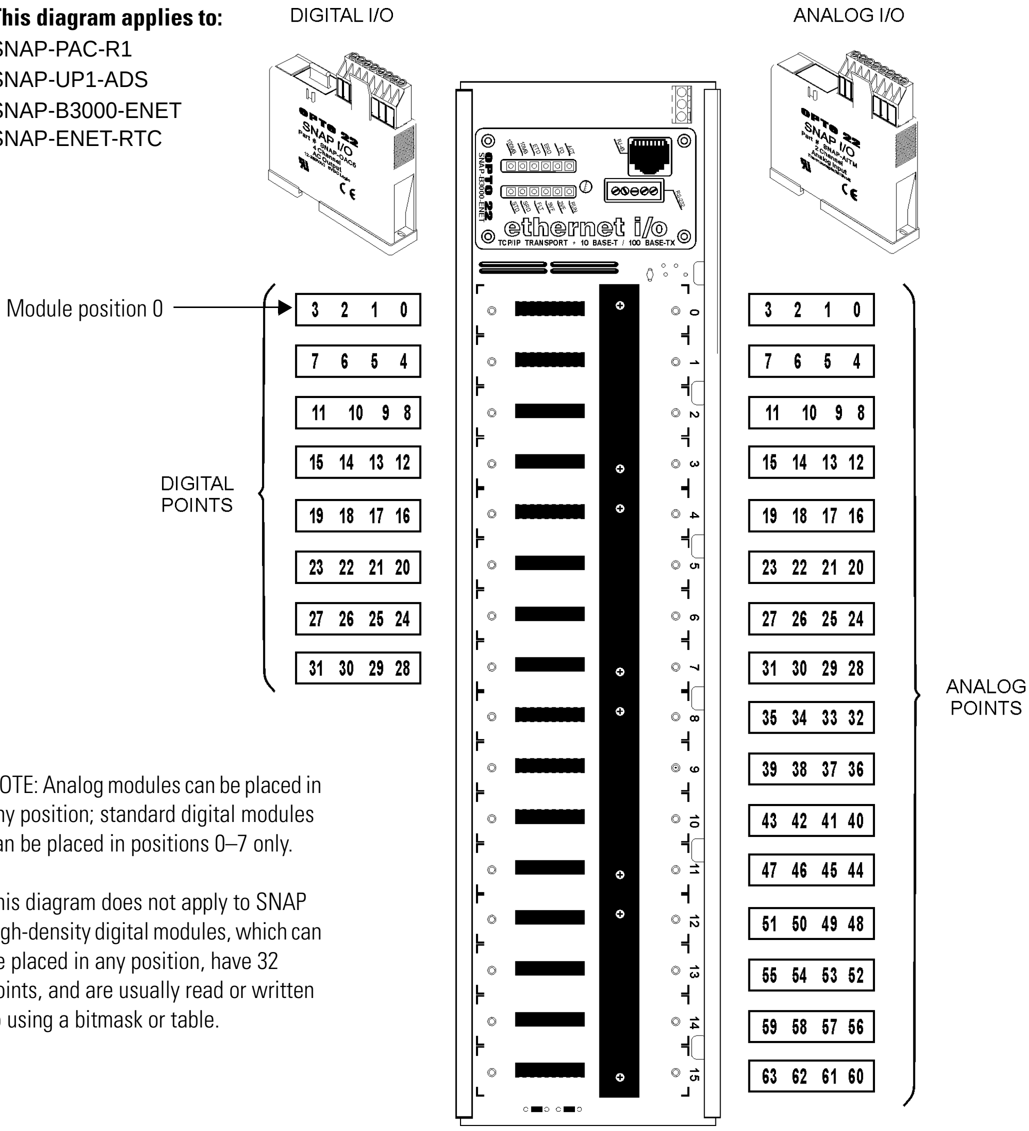
CAUTION: *Make certain you use a rack shown as compatible for the processor.* ***Using any other rack will severely damage the brain or controller.***

The mounting racks used with these processors can hold either 4, 8, 12, or 16 Opto 22 SNAP I/O® modules. Analog, serial, and high-density digital modules can be placed in any position on these racks. For the larger racks, standard digital modules can be placed in positions 0–7 only. For more information, see the data sheet for your rack. Data sheets can be downloaded from www.opto22.com

Each standard SNAP digital module contains four input or four output points. SNAP analog modules supported by these processors contain either two or four points.

Each point on the rack is numbered; when you configure the point or read or write to it, you reference it by its number. The following diagram shows the largest rack as an example. Note that four numbers are shown for each analog module. If you are using analog modules that contain only two points, ignore the upper two point numbers.

This diagram applies to:
SNAP-PAC-R1
SNAP-UP1-ADS
SNAP-B3000-ENET
SNAP-ENET-RTC



NOTE: Analog modules can be placed in any position; standard digital modules can be placed in positions 0–7 only.

This diagram does not apply to SNAP high-density digital modules, which can be placed in any position, have 32 points, and are usually read or written to using a bitmask or table.

SNAP Ethernet Analog and Simple Digital Systems

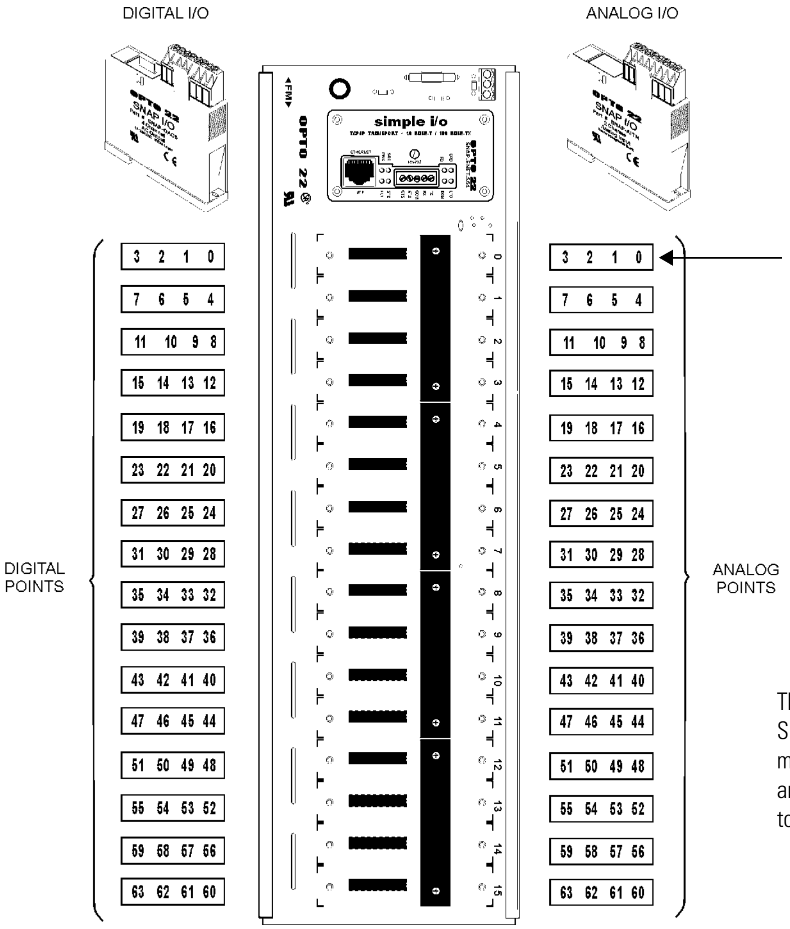
Processors	Compatible Racks
SNAP-PAC-R2 SNAP-UP1-M64 SNAP-ENET-S64	SNAP-M series (SNAP-M16, SNAP-M32, SNAP-M48, SNAP-M64)

CAUTION:** Make certain you use a rack shown as compatible for the processor.* ***Using any other rack will severely damage the brain or controller.

SNAP M-series mounting racks can hold up to 4, 8, 12, or 16 Opto 22 SNAP I/O modules. Any combination of analog, digital, serial, and high-density modules can be placed in any position on the rack (not exceeding eight serial modules).

Each standard SNAP digital module contains four input or four output channels (points). Digital functions on this brain are limited; see the brain's data sheet for specifications. SNAP analog modules supported by these processors contain either two or four points.

Each point on the rack is numbered; when you configure the point, you reference it by its number. The following diagram shows the largest rack as an example. All modules start with the same point numbers in position zero on the rack. If you are using analog modules with only two points, the top two addresses for those analog modules will be empty.



This diagram applies to:

SNAP-PAC-R2
SNAP-UP1-M64
SNAP-ENET-S64

This diagram does not apply to SNAP high-density digital modules, which have 32 points and are usually read or written to using a bitmask or table.

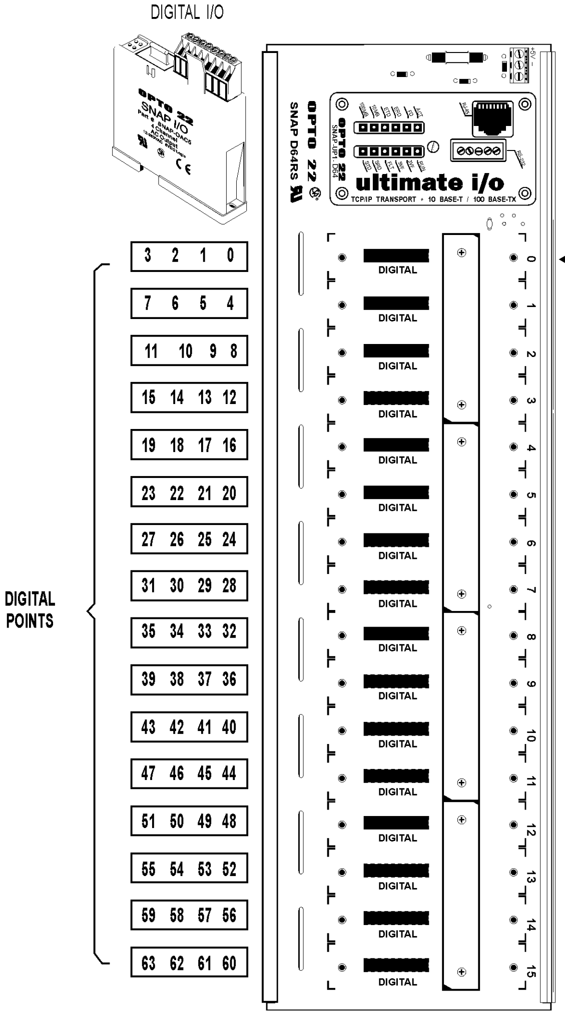
SNAP Ethernet Digital-Only Systems

Processors	Compatible Racks
SNAP-UP1-D64 SNAP-ENET-D64	SNAP-D64RS

CAUTION: *Do NOT connect a digital-only processor to any rack except the SNAP-D64RS.* ***Using any other rack will severely damage the brain or controller.***

A SNAP-D64RS I/O mounting rack can hold up to 16 standard digital modules. Analog, serial, and high-density digital modules *cannot* be used with digital-only processors. Each SNAP standard digital module contains four input or four output channels, for a total of 64 points of I/O on the rack.

Each point on the rack is numbered; when you configure the point, you reference it by its number. The diagram below shows the reference numbers for all 64 digital points.



This diagram applies to:
SNAP-UP1-D64
SNAP-ENET-D64

This diagram does not apply to SNAP high-density digital modules, which cannot be used with digital-only processors.

SNAP Serial-Based (mistic) I/O Units

Brains	Racks
B3000 (serial) SNAP-BRS	SNAP-B series (SNAP-BRS is limited to 8-module racks)

SNAP B-series mounting racks can hold either 4, 8, 12, or 16 Opto 22 SNAP I/O modules. The serial B3000 brain can use any size B-series rack and both analog and digital modules. It cannot use serial or high-density digital modules. The SNAP-BRS uses standard SNAP digital modules only (no high-density digital) and only on an 8-module rack.

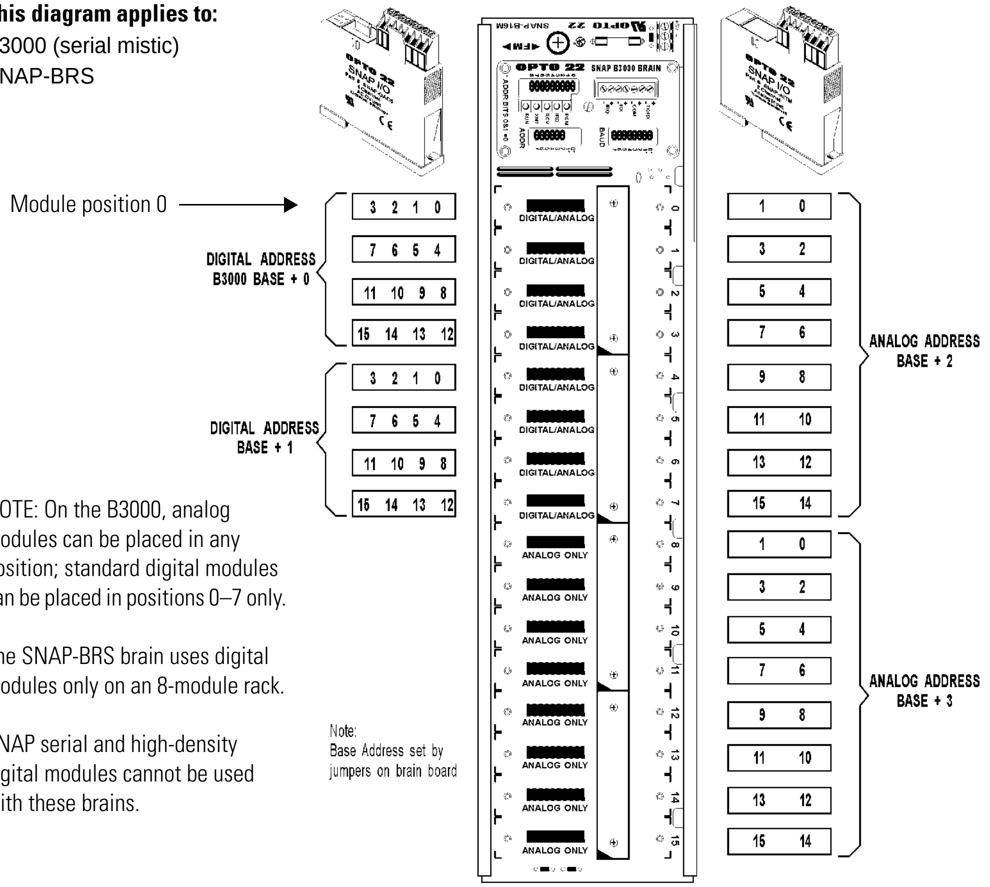
Analog modules can be placed in any position on these racks. For the larger racks, standard SNAP digital modules can be placed in positions 0–7 only. Each standard SNAP digital module contains four input or four output points. SNAP analog modules supported by the B3000 contain either one or two points.

Each point on the rack is numbered; when you configure the point or read or write to it, you reference it by its number. Each SNAP brain is really up to four logical brains in one: two digital 16-channel multifunctional brains plus (for a B3000) up to two analog 16-channel multifunctional brains.

Jumpers on the SNAP brain set the base address, which must be zero, four, or a multiple of four. The base address is the first digital address, digital channels 0–15. The second digital address, digital channels 16–31, is the base address plus one. On a B3000, the first analog address, analog channels 0–15, is the base address plus two; the second analog address, analog channels 16–31, is the base address plus three.

The following diagram shows the largest rack as an example. Note that two numbers are shown for each analog module. If you are using analog modules that contain only one point, ignore the upper point number.

This diagram applies to:
B3000 (serial mistic)
SNAP-BRS



NOTE: On the B3000, analog modules can be placed in any position; standard digital modules can be placed in positions 0–7 only.

The SNAP-BRS brain uses digital modules only on an 8-module rack.

SNAP serial and high-density digital modules cannot be used with these brains.

Non-SNAP Serial-Based (mistic) I/O Units

Brain boards	Racks
G4D16R G4D32RS G4A8R	Bricks (brain is built into rack)
B100 B200	PB16 G4PB16 PB16A

Because non-SNAP serial-based brain boards generally have a maximum of 16 modules on the rack, and each module has only one point, addressing points is simple. Point numbers correspond to module positions on the rack, 0–15, except for the following:

- If you are using Quad Pak modules, each module contains four points: module position zero contains points 0–3, module position one contains points 4–7, and so on.

- A G4D32RS brick contains the equivalent of two 16-module units, and the brain board addresses them separately. When you configure it in ioControl, notice that it is called a G4D16RS. Configure two G4D16RS I/O units for each brick.

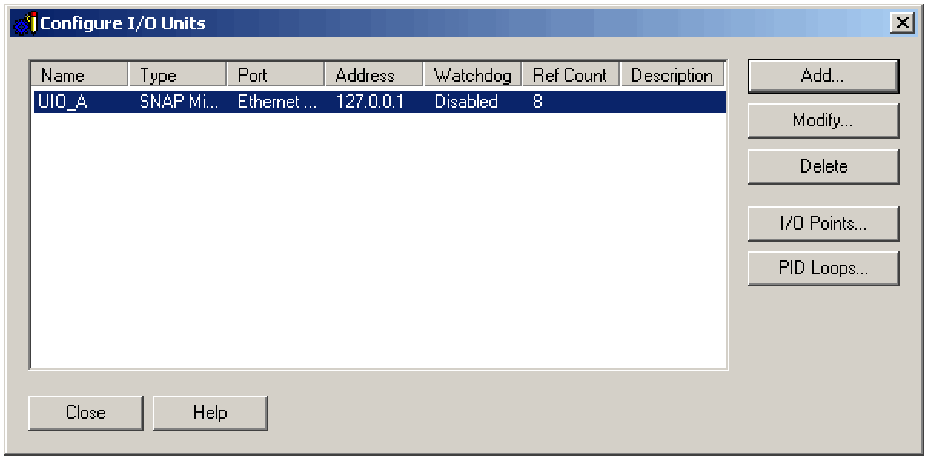
Adding an I/O Unit

NOTE: If you have already configured I/O in ioManager, you can also add I/O units or points if necessary from within ioControl, as you add commands to the blocks in your strategy.

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, double-click the I/O Units folder.

   You can also click the Configure I/O icon on the toolbar, or select Configure→I/O.

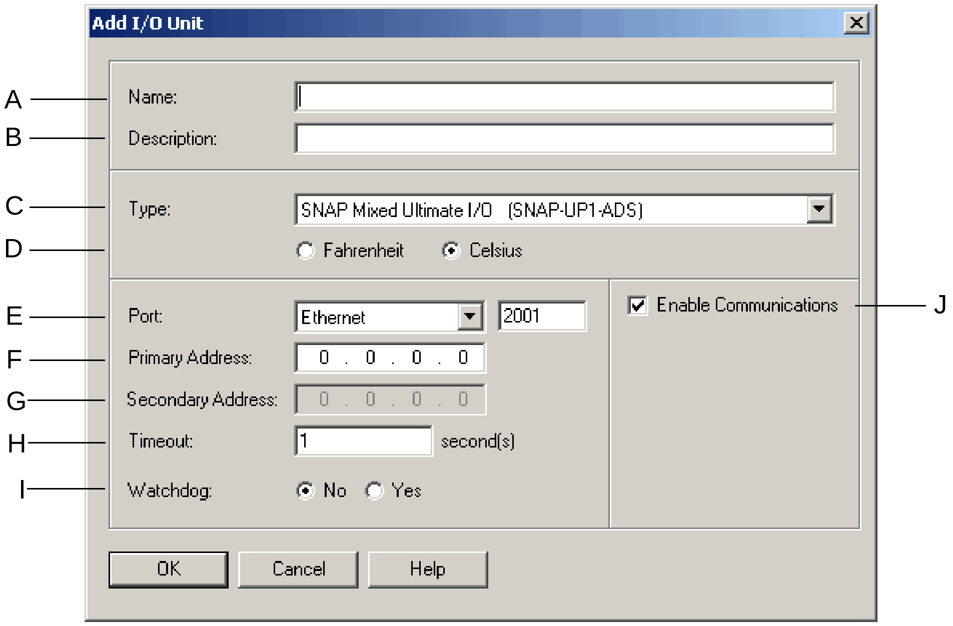
   The Configure I/O Units dialog box opens, showing all configured I/O units.



Name	Type	Port	Address	Watchdog	Ref Count	Description
UIO_A	SNAP Mi...	Ethernet ...	127.0.0.1	Disabled	8	

2. To configure a new I/O unit, click Add or double-click anywhere in the box below any listed units.

The Add I/O Unit dialog box appears.



Add I/O Unit

Name:

Description:

Type: SNAP Mixed Ultimate I/O (SNAP-UP1-ADS)

○ Fahrenheit ● Celsius

Port: Ethernet 2001

Primary Address: 0 . 0 . 0 . 0

Secondary Address: 0 . 0 . 0 . 0

Timeout: 1 second(s)

Watchdog: ● No ○ Yes

☑ Enable Communications

3. Complete the fields as described in "Add I/O Unit Dialog Box" below.

Add I/O Unit Dialog Box

(A) Name Enter a name for the I/O unit. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

(B) Description (Optional) Enter a description of the unit.

(C) Type Select the type of I/O unit from the drop-down list. Brain model numbers are listed in parentheses to help you determine which type to choose.

(D) Fahrenheit/Celsius (Analog and Mixed units only) Choose whether temperatures will be handled in Fahrenheit or Celsius.

(E) Port For Ethernet-based I/O units, communication with the control engine is Ethernet, and the default port number is 2001. If you have changed this port for security purposes, also change it here. (See the controller's user guide for details.)

For serial-based I/O units, the communication port and parameters are shown for you. Binary/CRC settings on your hardware are required. If you need to change the baud rate, see "Changing the Baud Rate for Serial I/O Units."

(F) Primary Address For Ethernet I/O units, type the IP address of the brain attached to the I/O unit. If the unit is local (the same SNAP Ultimate brain on which the strategy will run), use `127.0.0.1` as the IP address. This is a loopback address; it tells the brain to talk to itself, so if you change the brain's IP address, you don't have to change the address for the I/O unit.

For serial units, type the unit's address (valid range is 0–255).

(G) Secondary Address (Optional—ioControl Professional only—Ethernet-based I/O units only) To designate a secondary I/O unit for communication if this I/O unit is unavailable, enter the secondary unit's IP address. Note that both I/O units use the same port number.

(H) Timeout Enter the length of time the control engine should wait for a response when communicating with this I/O unit. Default is 1 second. **CAUTION:** For Ethernet, two retries are built in. If the timeout is long and the I/O unit is turned off or unreachable, the control engine could take quite a while to execute a command that talks to I/O.

(I) Watchdog Select whether you want a Watchdog on the unit (not available on remote simple I/O units). The default is No (disabled). If you select Yes, a new field appears; enter the Watchdog timeout value (in seconds). The default timeout is 0.5 seconds.

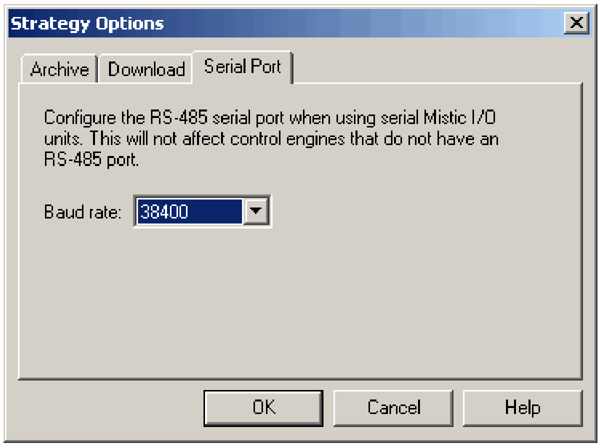
With a Watchdog, the I/O unit monitors activity on the port. If no communication is received for the specified interval, the unit assumes a Watchdog state. All selected outputs will then immediately go to a specified value, as configured in the Watchdog field of the Add Analog Point or Add Digital Point dialog boxes. (See "Adding I/O Points" on page 6-16.)

(J) Enable Communications Select whether you want communication to the I/O unit enabled or disabled on startup. Enabled is the default. Disabling communication to an I/O unit is the same as disabling communication to all points on the unit.

Changing the Baud Rate for Serial I/O Units

If the default baud rate shown in the Add I/O Unit dialog box not correct, you can change it. This baud rate is for communication through the RS-485 port to serial I/O units. It does not affect other serial ports on the controller.

1. Make sure the strategy is open and in Configure mode.
2. From the File menu, choose Strategy Options. Click the Serial Port tab.



Configure the RS-485 serial port when using serial Mistic I/O units. This will not affect control engines that do not have an RS-485 port.

Baud rate: 38400

3. Choose the correct baud rate from the drop-down list and click OK.

   The baud rate to serial I/O units is changed.

Changing Configured I/O Units

1. To change a configured I/O unit, make sure the strategy is open and in Configure mode.
2. Find the I/O unit's name on the Strategy Tree. Double-click it to open the Edit I/O Unit dialog box.
3. Make the necessary changes and click OK.

You can also change an I/O unit from the Configure I/O Units dialog box by double-clicking the unit's name or highlighting it and clicking Modify.

Deleting Configured I/O Units

You cannot delete an I/O unit if it has I/O points configured or if the I/O unit is referenced in an ioControl command.

CAUTION: *Be careful when deleting I/O units. You cannot undo a deletion.*

1. To delete a configured I/O unit, make sure the strategy is open and in Configure mode.
2. Find the I/O unit's name on the Strategy Tree. Right-click it and choose Delete from the pop-up menu.

   The I/O unit is deleted from the strategy.

You can also delete an I/O unit from the Configure I/O Units dialog box by highlighting the unit's name and clicking Delete.

Adding I/O Points

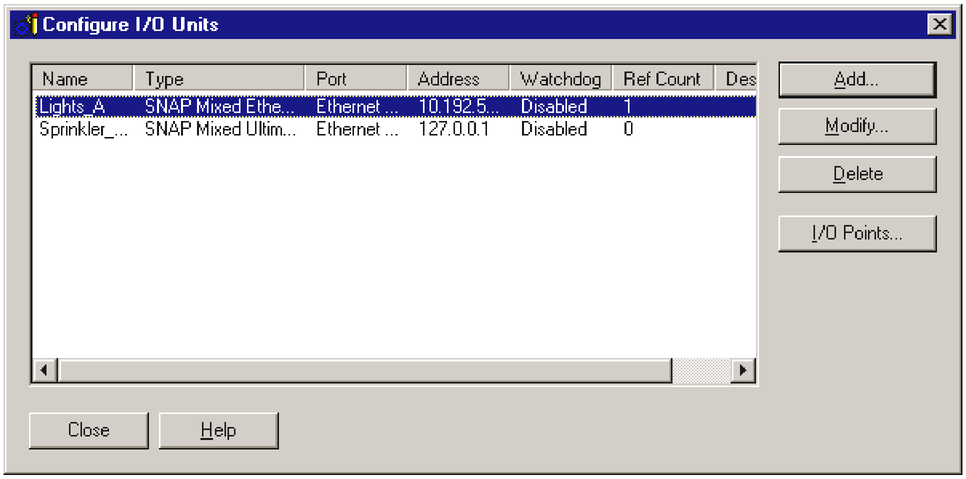
Before you add an individual I/O point, such as a sensor or a switch, you must add the I/O unit the point is on. See "Adding an I/O Unit" on page 6-12.

Adding a Digital I/O Point

NOTE: This section applies to points on most digital modules, but not to points on SNAP high-density digital modules. For information on high-density digital modules, see page 6-25.

1. With the strategy open and in Configure mode, double-click the I/O Units folder (not the individual unit's icon) on the Strategy Tree.

The Configure I/O Units dialog box appears.



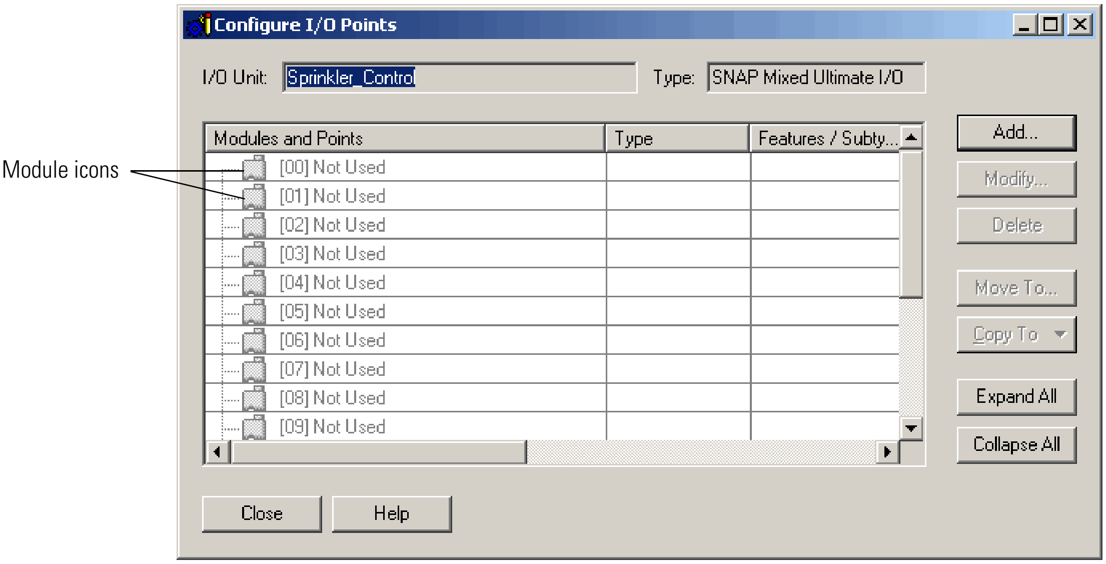
Name	Type	Port	Address	Watchdog	Ref Count	Des
Lights_A	SNAP Mixed Ethe...	Ethernet ...	10.192.5....	Disabled	1	
Sprinkler_...	SNAP Mixed Ultim...	Ethernet ...	127.0.0.1	Disabled	0	

2. Highlight the I/O unit the points are on, and click I/O Points.

   The Configure I/O Points dialog box appears.

3. (ioControl Pro only) **If you are using a non-SNAP I/O unit,** double-click the channel you want to use. Skip to step 8.

4. **If you are using a SNAP I/O unit,** notice the module icons in the dialog box.

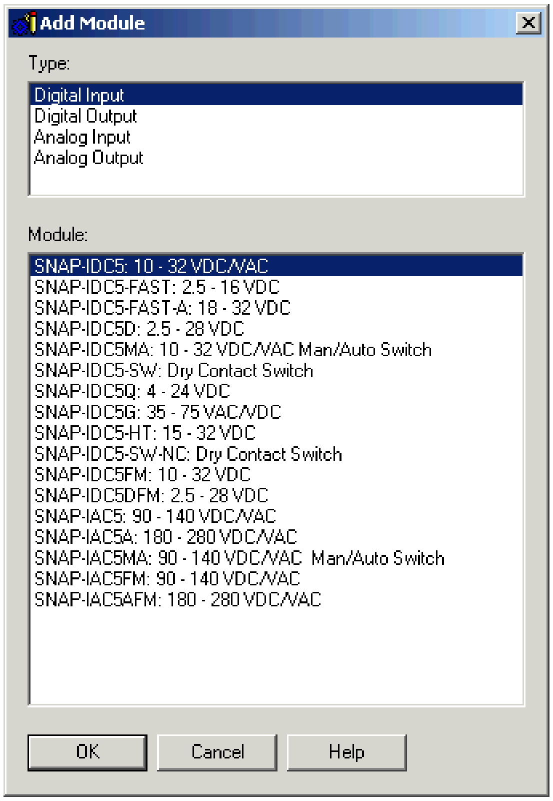


I/O Unit: Sprinkler_Control   Type: SNAP Mixed Ultimate I/O

Modules and Points	Type	Features / Subty...
[00] Not Used		
[01] Not Used		
[02] Not Used		
[03] Not Used		
[04] Not Used		
[05] Not Used		
[06] Not Used		
[07] Not Used		
[08] Not Used		
[09] Not Used		

NOTE: On an Ethernet SNAP I/O unit, you can configure all the analog and digital modules on the rack at once. On a serial SNAP I/O unit, you must configure digital and analog modules separately.

5. Highlight the number that represents the module's position on the rack. (See the diagrams in "Addressing I/O Units" on page 6-6.) Click Add.



Add Module

Type:

Digital Input
Digital Output
Analog Input
Analog Output

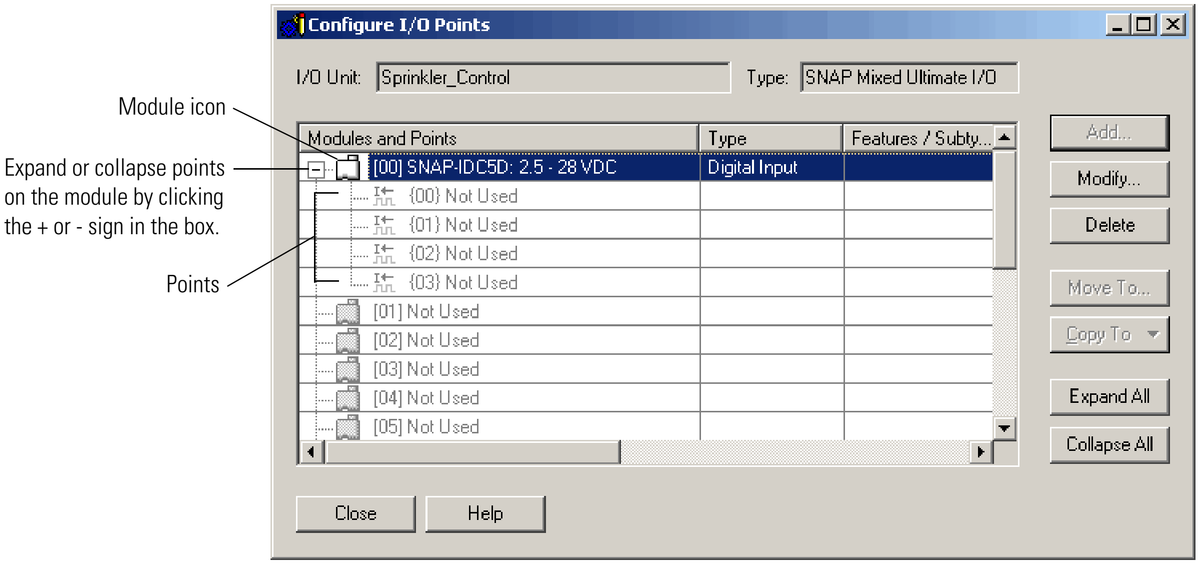
Module:

SNAP-IDC5: 10 - 32 VDC/VAC
SNAP-IDC5-FAST: 2.5 - 16 VDC
SNAP-IDC5-FAST-A: 18 - 32 VDC
SNAP-IDC5D: 2.5 - 28 VDC
SNAP-IDC5MA: 10 - 32 VDC/VAC Man/Auto Switch
SNAP-IDC5-SW: Dry Contact Switch
SNAP-IDC5Q: 4 - 24 VDC
SNAP-IDC5G: 35 - 75 VAC/VDC
SNAP-IDC5-HT: 15 - 32 VDC
SNAP-IDC5-SW-NC: Dry Contact Switch
SNAP-IDC5FM: 10 - 32 VDC
SNAP-IDC5DFM: 2.5 - 28 VDC
SNAP-IAC5: 90 - 140 VDC/VAC
SNAP-IAC5A: 180 - 280 VDC/VAC
SNAP-IAC5MA: 90 - 140 VDC/VAC Man/Auto Switch
SNAP-IAC5FM: 90 - 140 VDC/VAC
SNAP-IAC5AFM: 180 - 280 VDC/VAC

6. In the Add Module dialog box, choose the module type and then the exact module from the lists. Click OK.

7. In the Configure I/O Points dialog box, click the plus sign next to the new module to expand it.

   Notice that the module icon is color-coded to reflect the type of module being configured: white for digital DC input, red for digital DC output, yellow for digital AC input, and black for digital AC output.



I/O Unit: Sprinkler_Control   Type: SNAP Mixed Ultimate I/O

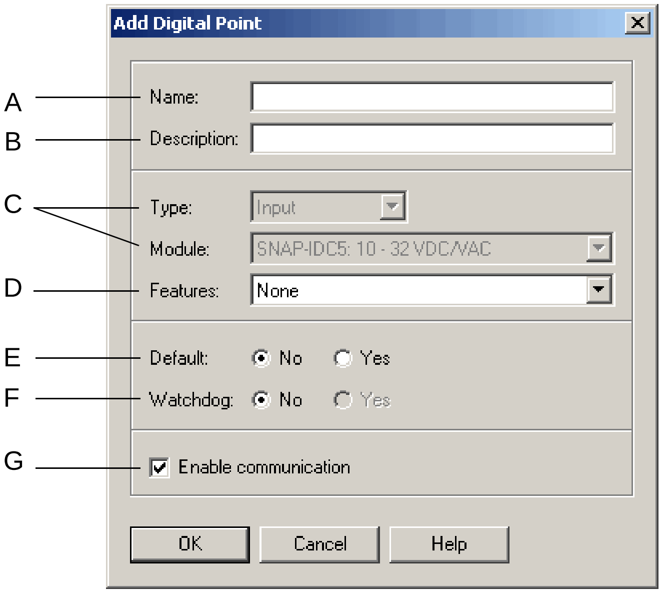
Modules and Points	Type	Features / Subty...
[00] SNAP-IDC5D: 2.5 - 28 VDC	Digital Input	
{00} Not Used		
{01} Not Used		
{02} Not Used		
{03} Not Used		
[01] Not Used		
[02] Not Used		
[03] Not Used		
[04] Not Used		
[05] Not Used		

Module icon

Expand or collapse points on the module by clicking the + or - sign in the box.

Points

8. Highlight the point you want to configure and click Add.



Name:

Description:

Type: Input

Module: SNAP-IDC5: 10 - 32 VDC/VAC

Features: None

Default: ● No ○ Yes

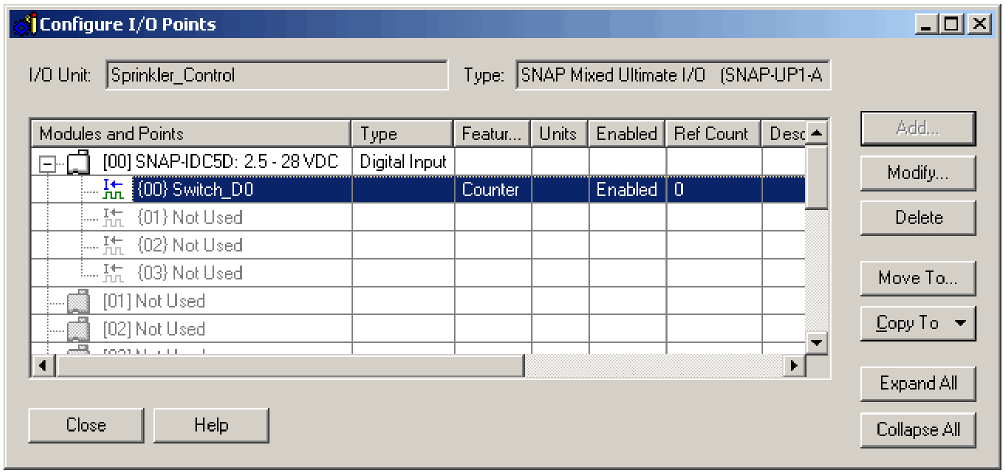
Watchdog: ● No ○ Yes

☑ Enable communication

9. Complete the fields as described in "Add Digital Point Dialog Box" below.

10. When you have completed the fields, click OK.

The new point appears in the list. Here is an example of how it might look on a SNAP I/O unit:



I/O Unit: Sprinkler_Control Type: SNAP Mixed Ultimate I/O (SNAP-UP1-A

Modules and Points	Type	Featur...	Units	Enabled	Ref Count	Desc
[00] SNAP-IDC5D: 2.5 - 28 VDC	Digital Input					
{00} Switch_D0		Counter		Enabled	0	
{01} Not Used						
{02} Not Used						
{03} Not Used						
[01] Not Used						
[02] Not Used						

NOTE: If you need to add several similar points, see "Copying a Configured I/O Point" on page 6-26.

Add Digital Point Dialog Box

(A) Name Enter a name for the point. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

(B) Description (Optional) Enter a description of the point.

(C) Type/Module For non-SNAP I/O units, choose the module type and the exact module from the drop-down lists.

For SNAP I/O units: Type and module are already filled in for you, as shown in the example above.

(D) Features To use a feature of the module, choose it from the drop-down list. You can configure some input modules with a counter, totalizer, or other feature. (Inputs automatically have both on-latches and off-latches.)

(E) Default To set a default state for the point when the strategy is run, click Yes and choose the state (Off or On). To leave the point in the state it was before, click No.

(F) Watchdog (Output modules only) To set a Watchdog, click Yes and choose On or Off from the drop-down list.

(G) Enable Communication Select whether you want communication to this I/O point enabled or disabled on startup. Enabled is the default.

Adding an Analog I/O Point

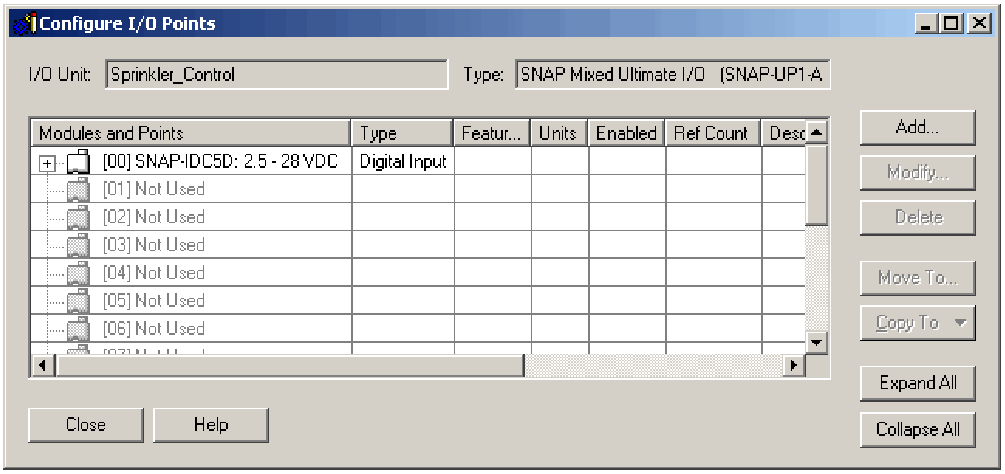
1. With the strategy open and in Configure mode, double-click the I/O Units folder (not the individual unit's icon) on the Strategy Tree.

2. In the Configure I/O Units dialog box, highlight the I/O unit the points are on, and click I/O Points.

   The Configure I/O Points dialog box appears.

3. (ioControl Pro only) **If you are using a non-SNAP I/O unit,** double-click the channel you want to use. Skip to step 8.

4. **If you are using a SNAP I/O unit**, notice the module icons in the Configure I/O Points dialog box.



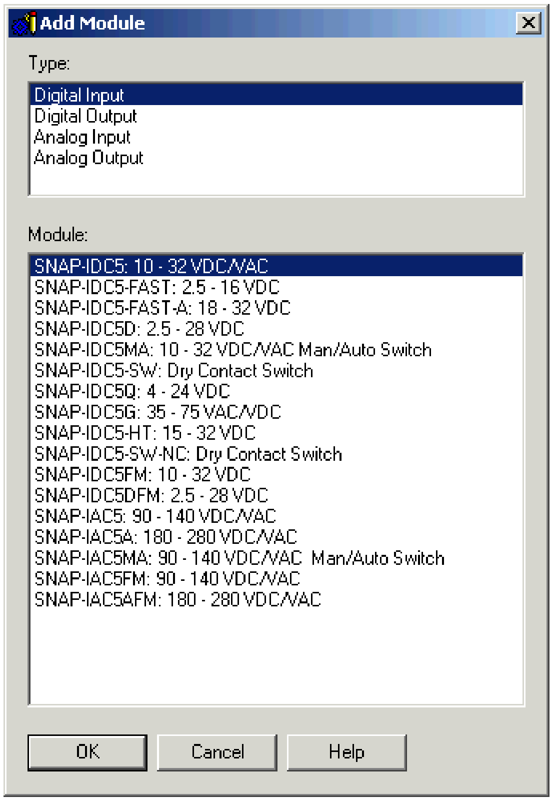
I/O Unit: Sprinkler_Control

Type: SNAP Mixed Ultimate I/O (SNAP-UP1-A

Modules and Points	Type	Featur...	Units	Enabled	Ref Count	Desc
[00] SNAP-IDC5D: 2.5 - 28 VDC	Digital Input					
[01] Not Used						
[02] Not Used						
[03] Not Used						
[04] Not Used						
[05] Not Used						
[06] Not Used						

NOTE: In this example, a digital module has already been added in position zero. This example shows an Ethernet SNAP I/O unit, so both digital and analog modules can be configured at the same time. On a serial SNAP I/O unit, the digital module appears, but digital and analog modules must be configured separately.

5. Highlight the number of the analog module's position on the rack. (See the diagrams in "Addressing I/O Units" on page 6-6.) Click Add.



Type:

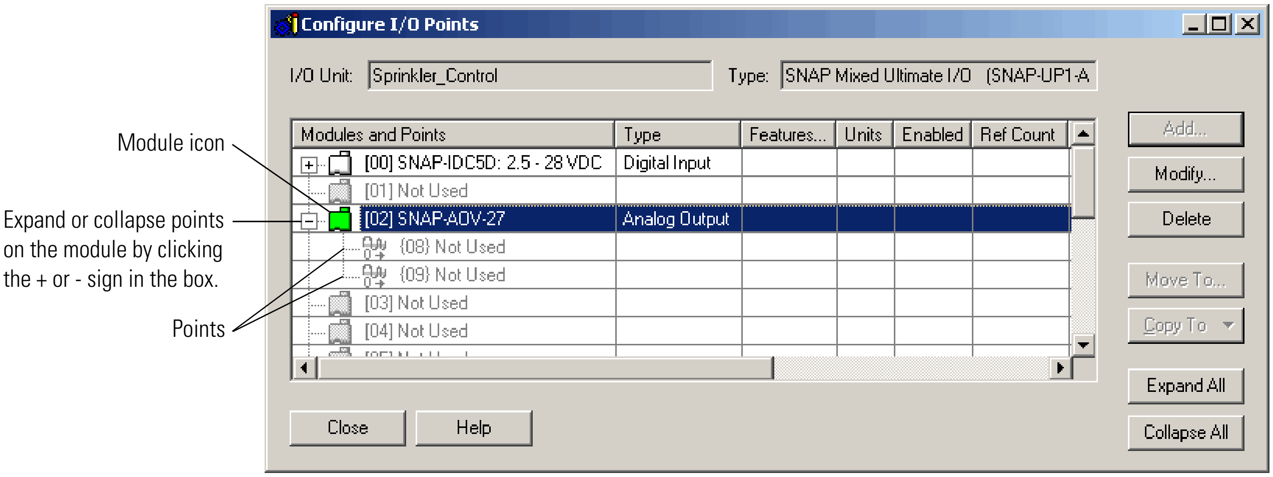
- Digital Input
- Digital Output
- Analog Input
- Analog Output

Module:

- SNAP-IDC5: 10 - 32 VDC/VAC
- SNAP-IDC5-FAST: 2.5 - 16 VDC
- SNAP-IDC5-FAST-A: 18 - 32 VDC
- SNAP-IDC5D: 2.5 - 28 VDC
- SNAP-IDC5MA: 10 - 32 VDC/VAC Man/Auto Switch
- SNAP-IDC5-SW: Dry Contact Switch
- SNAP-IDC5Q: 4 - 24 VDC
- SNAP-IDC5G: 35 - 75 VAC/VDC
- SNAP-IDC5-HT: 15 - 32 VDC
- SNAP-IDC5-SW-NC: Dry Contact Switch
- SNAP-IDC5FM: 10 - 32 VDC
- SNAP-IDC5DFM: 2.5 - 28 VDC
- SNAP-IAC5: 90 - 140 VDC/VAC
- SNAP-IAC5A: 180 - 280 VDC/VAC
- SNAP-IAC5MA: 90 - 140 VDC/VAC Man/Auto Switch
- SNAP-IAC5FM: 90 - 140 VDC/VAC
- SNAP-IAC5AFM: 180 - 280 VDC/VAC

6. In the Add Module dialog box, choose the module type and then the exact module from the lists. Click OK.

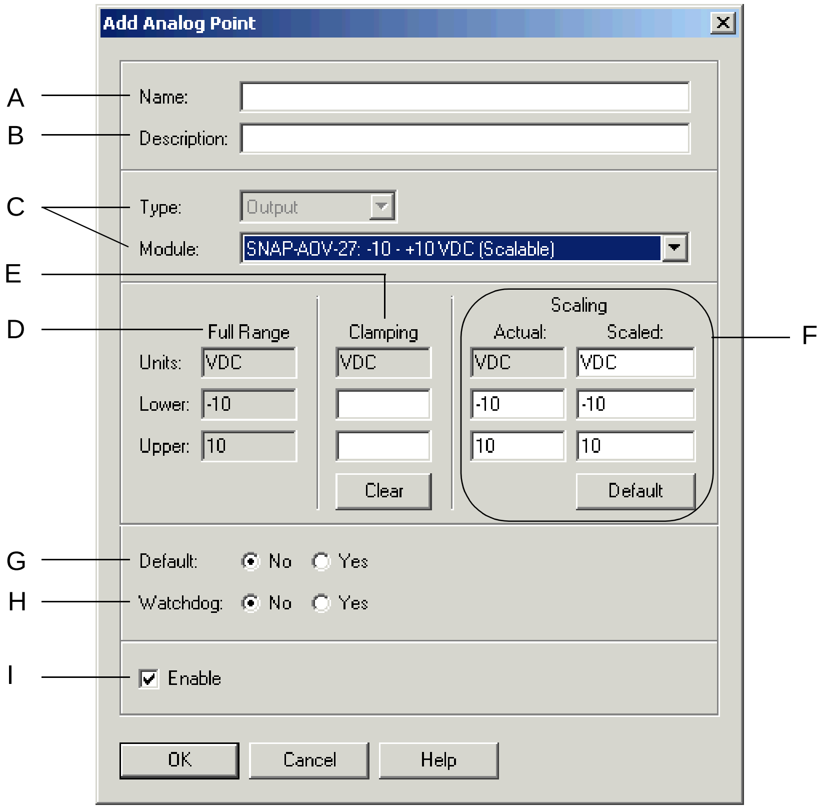
7. In the Configure I/O Points dialog box, click the plus sign next to the new module to expand it. Notice that the module icon is color-coded to reflect the type of module being configured: blue for analog input, green for analog output.


Module icon
Expand or collapse points on the module by clicking the + or - sign in the box.
Points
Configure I/O Points

I/O Unit: Sprinkler_Control Type: SNAP Mixed Ultimate I/O (SNAP-UP1-A

Modules and Points	Type	Features...	Units	Enabled	Ref Count
[00] SNAP-IDC5D: 2.5 - 28 VDC	Digital Input				
[01] Not Used					
[02] SNAP-AOV-27	Analog Output				
{08} Not Used					
{09} Not Used					
[03] Not Used					
[04] Not Used					

8. Highlight the point you want to configure and click Add.



Add Analog Point

Name:

Description:

Type: Output

Module: SNAP-AOV-27: -10 - +10 VDC (Scalable)

	Full Range	Clamping	Scaling Actual:	Scaling Scaled:
Units:	VDC	VDC	VDC	VDC
Lower:	-10		-10	-10
Upper:	10		10	10

Default: ● No ○ Yes

Watchdog: ● No ○ Yes

☑ Enable

9. Complete the fields as described in "Add Analog Point Dialog Box" below.

10. When you have completed the fields, click OK.

The new point is added.

NOTE: If you need to add several similar points, see "Copying a Configured I/O Point" on page 6-26.

Add Analog Point Dialog Box

(A) Name Enter a name for the point. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

(B) Description (Optional) Enter a description of the point.

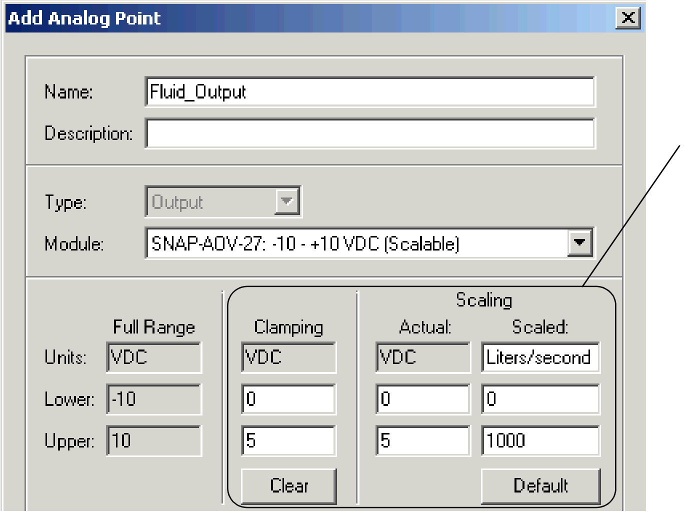
(C) Type/Module For non-SNAP I/O units, choose the module type and the exact module from the drop-down lists.

For SNAP I/O units: Type and module are already filled in for you. You may be able to choose a different range or a scalable module from the drop-down list.

(D) Full Range Full range and units for this module. If the module is scalable, use F to change scale.

(E) Clamping (Outputs only—optional) Enter upper and lower clamp if necessary to limit output to the device attached to the point. If fields are left empty, no clamp is applied. To empty the fields, click Clear.

(F) Scaling (Scalable modules only—optional) Use to assign custom units and values to the module. For example, you could scale the readings of a -10 to +10 VDC input point to measure its input as zero liters per second when the real-world reading is zero VDC, and 1000 liters per second when the real-world reading is five VDC. This example would look like this:



Name: Fluid_Output

Description:

Type: Output

Module: SNAP-AOV-27: -10 - +10 VDC (Scalable)

	Full Range	Clamping	Scaling Actual:	Scaling Scaled:
Units:	VDC	VDC	VDC	Liters/second
Lower:	-10	0	0	0
Upper:	10	5	5	1000

In this example units are changed to liters/second and lower and upper values are changed. Although the module has an output of -10 to +10 volts, the device attached to the point outputs only 0–5 volts. Scaling reflects the device's range.
In this case Clamping protects the device by ensuring that out-of-range voltage will never be sent to it.

Custom scaled values can be any floating point value as long as the upper value is higher than the lower value. Note that inputs typically have under-range and over-range capability, which means you can specify a lower or upper value beyond the standard value. Outputs do not have under-range or over-range capability.

To return the units and upper/lower values to the defaults for the module, click Default.

(G) Default To set the initial values for this I/O point when the strategy is run, click Yes and define the value. To leave the internal/external values at their last state, click No.

If communication to the point is disabled, only the internal values (IVALs) are updated by the control engine. The real-world external values (XVALs) for inputs don't change because they are controlled by external devices. The IVALs for inputs are set and will be overwritten by the XVALs unless communication to the point is disabled.

(H) Watchdog (Outputs only) To set a Watchdog on this point, click Yes, and define the value to be assigned to the output if the Watchdog is triggered. A Watchdog is triggered if no communication activity is detected on the bus for the amount of time specified in the Watchdog field of this point's I/O unit. For no Watchdog, click No.

(I) Enable Select whether you want communication to this I/O point enabled or disabled on startup. Enabled is the default.

Configuring Special-Purpose Modules

Configuring a Serial Module

Use ioManager to configure SNAP serial communication modules, including the Profibus module. Serial modules can be used with SNAP Ethernet-based I/O units only. See the *ioManager User's Guide* (Opto 22 form #1440) for instructions.

Configuring a SNAP High-Density Digital Module

SNAP high-density digital (HDD) modules do not require configuration because the I/O unit recognizes them automatically. Be sure to remember which module positions on the rack are occupied by HDD modules.

HDD modules are supported by SNAP Ethernet-based I/O units only. For more information on using these modules in your strategy, see "High-Density Digital Module Commands" on page 10-6.

Changing Point Configuration

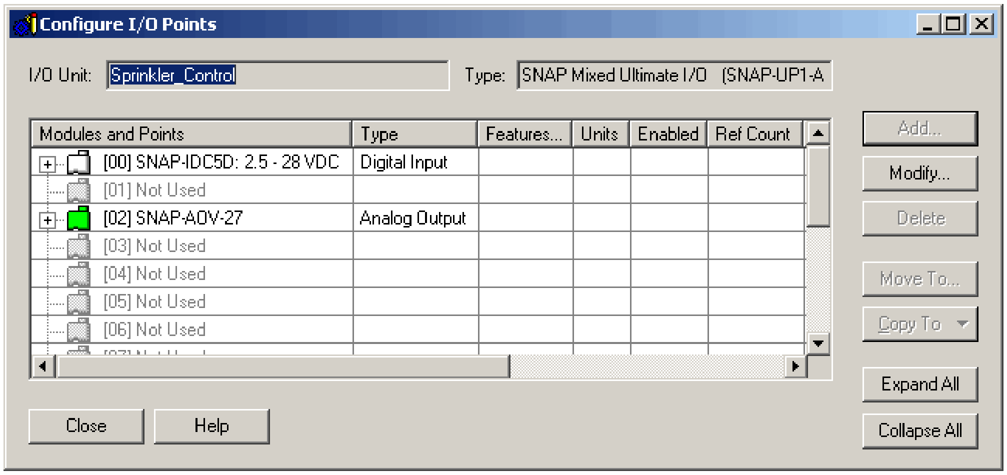
Moving a Configured I/O Point

You can move most configured I/O points to an empty point on the same I/O unit or on a different unit.

1. With the strategy open in Configure mode, double-click the I/O Units folder on the Strategy Tree.

2. In the Configure I/O Unit dialog box, highlight the unit the point is on and click I/O Points.

The Configure I/O Points dialog box opens. For a SNAP I/O unit, it looks something like this:

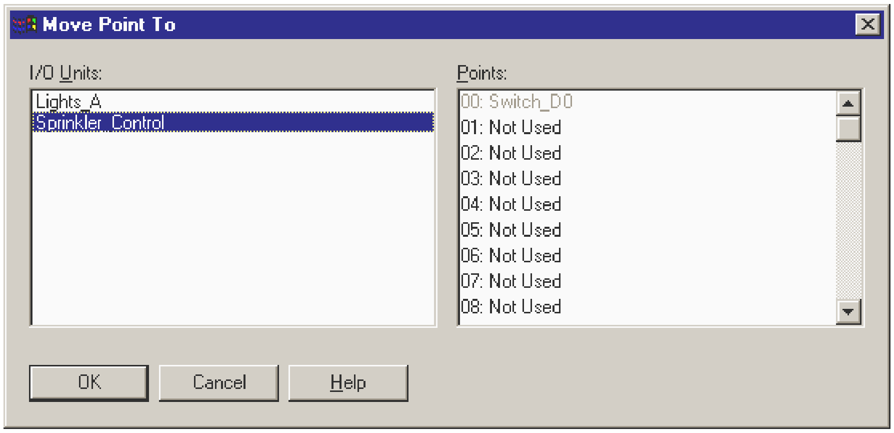


I/O Unit: Sprinkler_Control Type: SNAP Mixed Ultimate I/O (SNAP-UP1-A

Modules and Points	Type	Features...	Units	Enabled	Ref Count
[00] SNAP-IDC5D: 2.5 - 28 VDC	Digital Input				
[01] Not Used					
[02] SNAP-AOV-27	Analog Output				
[03] Not Used					
[04] Not Used					
[05] Not Used					
[06] Not Used					

3. If necessary, expand the modules by clicking Expand All.

4. Highlight the point you want to move and click Move To.



I/O Units:
Lights_A
Sprinkler_Control

Points:
00: Switch_D0
01: Not Used
02: Not Used
03: Not Used
04: Not Used
05: Not Used
06: Not Used
07: Not Used
08: Not Used

5. In the Points area of the Move Point To dialog box, highlight the location you are moving the point to. Then click OK.

   You return to the Configure I/O Points dialog box, and the point has been moved.

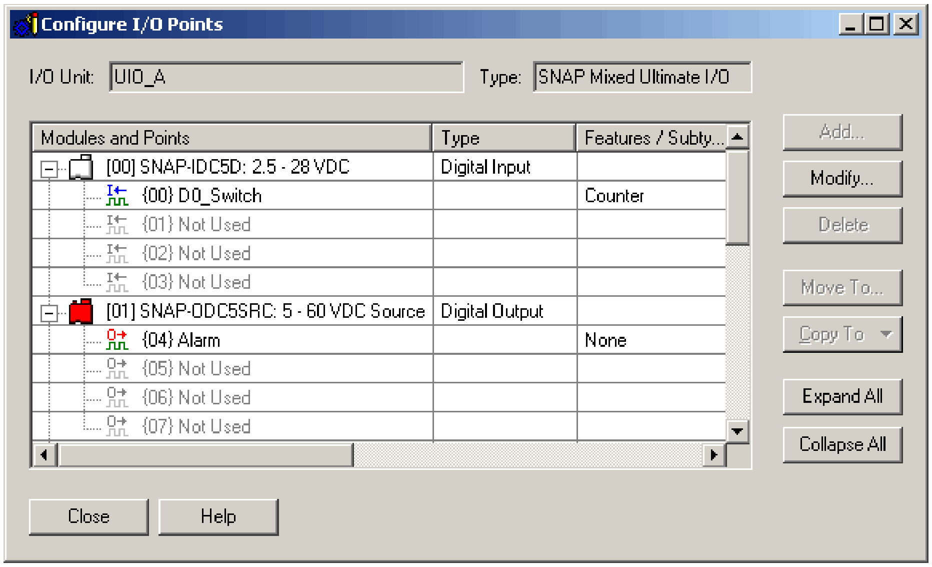
Copying a Configured I/O Point

If you have several points that are the same, you can copy a configured point:

- to fill empty points on the same module
- to fill all empty points on the I/O unit.
- to another other point on the same or a different I/O unit

1. With the strategy open in Configure mode, double-click the I/O Units folder on the Strategy Tree to open the Configure I/O Points dialog box.

2. Click Expand All so you can see the points.



Configure I/O Points

I/O Unit: UIO_A

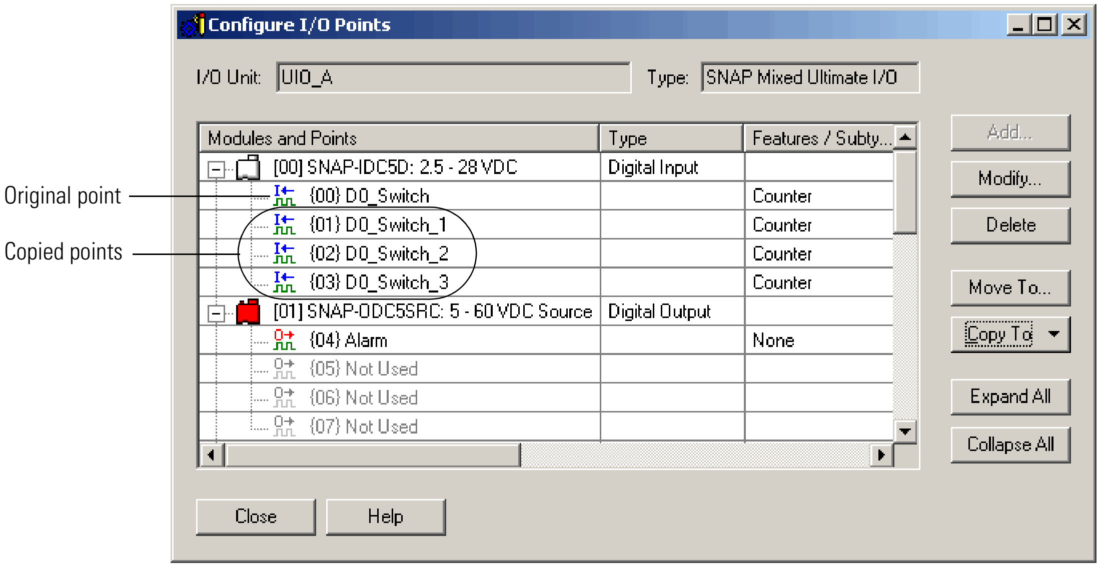
Type: SNAP Mixed Ultimate I/O

Modules and Points	Type	Features / Subty...
[00] SNAP-IDC5D: 2.5 - 28 VDC	Digital Input	
{00} DO_Switch		Counter
{01} Not Used		
{02} Not Used		
{03} Not Used		
[01] SNAP-ODC5SRC: 5 - 60 VDC Source	Digital Output	
{04} Alarm		None
{05} Not Used		
{06} Not Used		
{07} Not Used		

3. Highlight the point you want to copy and click the Copy To button. From the popup menu, choose one of the following:

To copy the point to fill empty points on the module, choose Fill In Module.

The point is copied to the other empty points on the same module.



Configure I/O Points

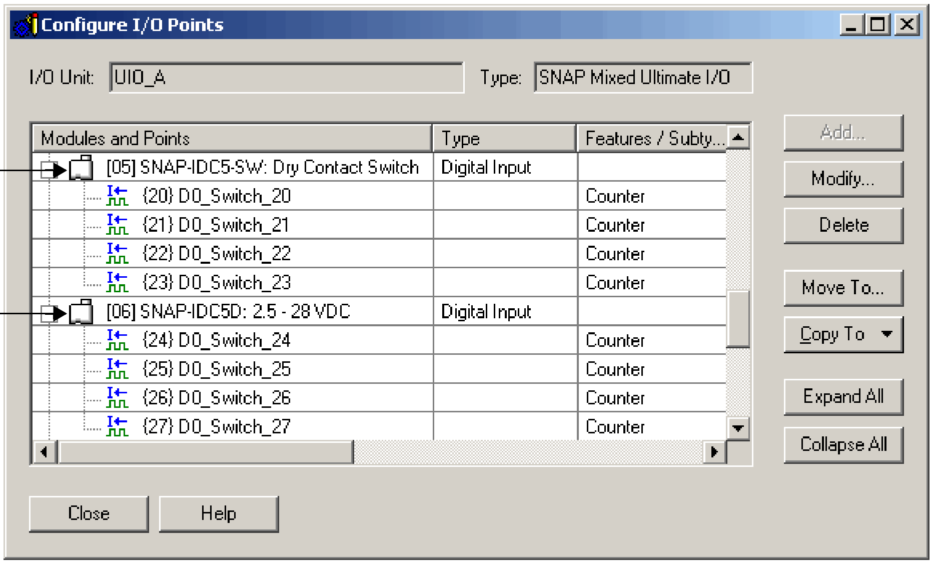
I/O Unit: UIO_A

Type: SNAP Mixed Ultimate I/O

Modules and Points	Type	Features / Subty...
[00] SNAP-IDC5D: 2.5 - 28 VDC	Digital Input	
{00} DO_Switch		Counter
{01} DO_Switch_1		Counter
{02} DO_Switch_2		Counter
{03} DO_Switch_3		Counter
[01] SNAP-ODC5SRC: 5 - 60 VDC Source	Digital Output	
{04} Alarm		None
{05} Not Used		
{06} Not Used		
{07} Not Used		

To copy the point to fill all empty points on the I/O unit, choose Fill In I/O Unit.

The point is copied to all empty points on similar modules (for example, to all empty points on digital input modules, whether they are the same or a different part number) and to all compatible slots without configured modules.



I/O Unit: UIO_A

Type: SNAP Mixed Ultimate I/O

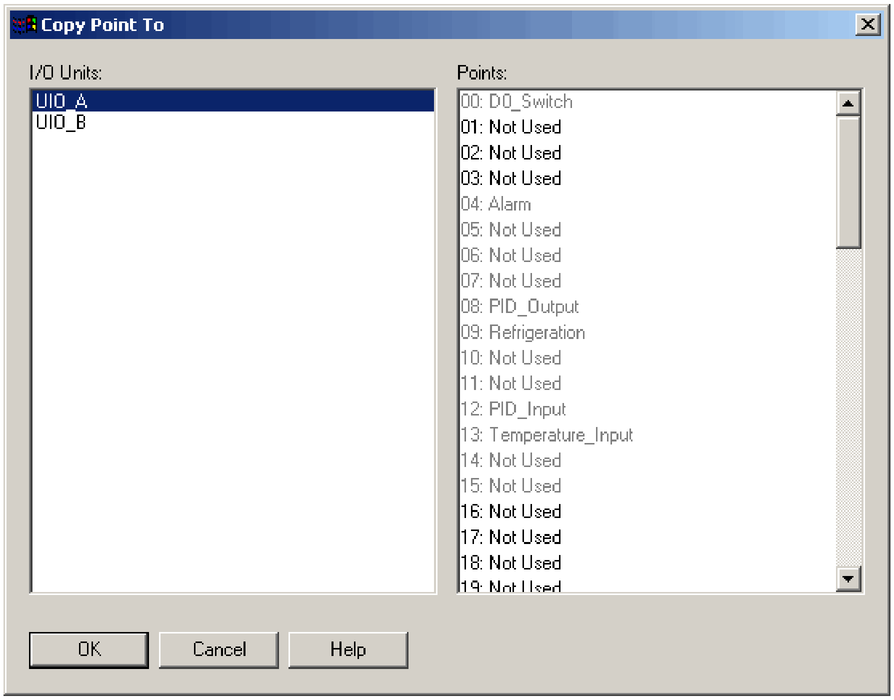
Modules and Points	Type	Features / Subty...
[05] SNAP-IDC5-SW: Dry Contact Switch	Digital Input	
{20} DO_Switch_20		Counter
{21} DO_Switch_21		Counter
{22} DO_Switch_22		Counter
{23} DO_Switch_23		Counter
[06] SNAP-IDC5D: 2.5 - 28 VDC	Digital Input	
{24} DO_Switch_24		Counter
{25} DO_Switch_25		Counter
{26} DO_Switch_26		Counter
{27} DO_Switch_27		Counter

This module was already configured. The copied point filled in all empty points, even though the module is a different part number.

No module existed in this slot. Both the points and the module were copied from the original.

To copy the point to one other point, either on the same or a different I/O unit, choose To Specific.

The Copy To dialog box opens.



I/O Units:

UIO_A
UIO_B

Points:

00: DO_Switch
01: Not Used
02: Not Used
03: Not Used
04: Alarm
05: Not Used
06: Not Used
07: Not Used
08: PID_Output
09: Refrigeration
10: Not Used
11: Not Used
12: PID_Input
13: Temperature_Input
14: Not Used
15: Not Used
16: Not Used
17: Not Used
18: Not Used
19: Not Used

In the left column, choose the same or another I/O unit. In the right column, choose the location of the point to copy to.

Available points are shown in black. Already configured points, points on a different type of module (digital output instead of digital input, for example), and point numbers not on the rack (such as point 40 on an 8-module rack) are grayed out.

If the point is copied to an empty module slot, the module part number from the copied point is added, too.

4. After you copy a point, change the new point as needed by double-clicking its point name. In the Edit Digital Point dialog box, type the new name and make any other changes.

Changing a Configured I/O Point

1. With the strategy open in Configure mode, expand the I/O Units folder on the Strategy Tree until you can see the I/O point you want to change. Double-click the I/O point name.

2. In the Edit Analog Point or Edit Digital Point dialog box, make the necessary changes. Click OK to save.

   For help in making changes, see the previous sections on adding I/O points.

Deleting a Configured I/O Point

You cannot delete an I/O point that is referred to in the Strategy.

CAUTION: *Be careful when deleting I/O points. You cannot undo a deletion.*

1. With the strategy open in Configure mode, expand the I/O Units folder on the Strategy Tree until you can see the I/O point you want to delete.

2. Right-click the I/O point's name and choose Delete from the pop-up menu.

You can also delete an I/O point from the Configure I/O Points dialog box by highlighting its name and clicking Delete.

Configuring PID Loops

PID loops (or simply PIDs) are used to drive an input (a process variable) toward a particular value (the setpoint) and keep the input very close to that value by controlling an output. For example, consider temperature control, where the input is a measurement of ambient temperature, the setpoint is the desired temperature, and the output is a heater. The PID for this system will use a mathematical formula that controls the output to maintain a desired temperature, efficiently adjust to changes in setpoint, and compensate for changes in load, such as the influx of cold air. In this example, a temperature sensor (analog input), a thermostat (analog input), and a heater control (analog output) are components of one system, controlled by a PID loop.

This guide assumes that you are already familiar with using PIDs. PID calculations are complex and the physical qualities of systems suitable for PID control differ greatly. This guide includes only basic information for configuring PIDs.

An analog/digital SNAP Ethernet I/O unit supports 16 PID loops; a SNAP Ultimate I/O unit supports 32 PID loops. If you have ioControl Professional, you can also use up to eight PID loops on mistic I/O units.

PIDs can control isolated systems or be part of cascaded systems where one loop controls the setpoints or input variables of others. For maximum flexibility, any PID input, setpoint, or output can be determined by ioControl commands.

PIDs and Strategies

Because PIDs run on the I/O unit, not the control engine, PIDs run whether the ioControl Strategy is running or not. Once running, a PID continues running until the I/O unit loses power or the PID is set to Manual.

When you change PID configuration in ioControl, remember that changes are not written to the I/O unit until the strategy is downloaded and run. For a SNAP Ethernet-based I/O unit, be sure to save PID configuration to flash memory following instructions for the I/O unit.

If you subsequently download a different strategy to the control engine, you'll receive an error message (-700) reminding you that a PID loop is still running and that it may conflict with the new strategy. To turn off a PID loop on a SNAP Ethernet-based I/O unit, open ioManager and use Inspect mode to change the PID's algorithm to None.

Each PID loop must be individually configured and tuned.

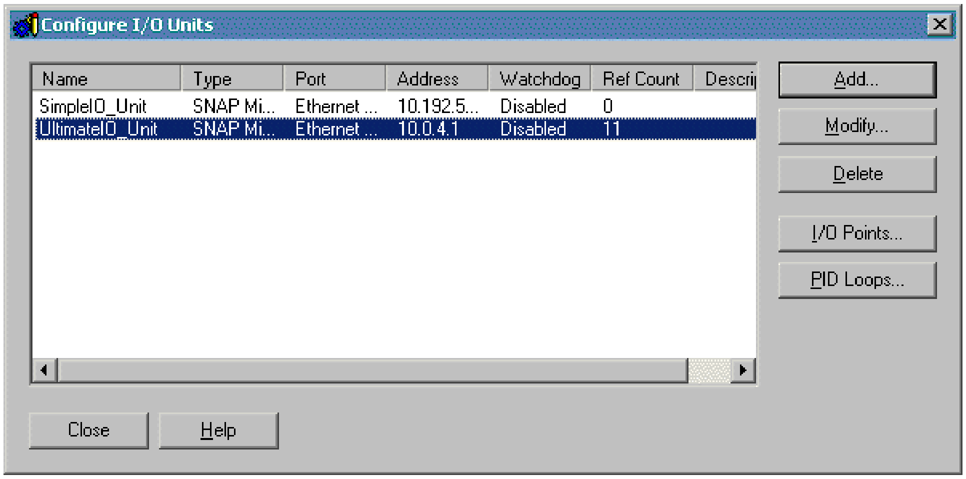
- **For SNAP Ethernet-based I/O units,** configuration steps start in the next section and tuning steps are described on page 6-58. For additional information, see "PID—Ethernet Commands" on page 10-58, and Opto 22 form #1410, *PID Configuration and Tuning: SNAP Ultimate I/O Learning Center Supplement*.
- (ioControl) **For serial-based mistic I/O units,** skip to "Adding a PID Loop (mistic)" on page 6-34. For additional information, see "PID—Mistic Commands" on page 10-62.

Adding a PID Loop (Ethernet)

NOTE: This section applies to SNAP Ethernet and SNAP Ultimate I/O units only.

1. With the strategy open and in Configure mode, double-click the I/O Units folder (not the individual unit's icon) on the Strategy Tree.

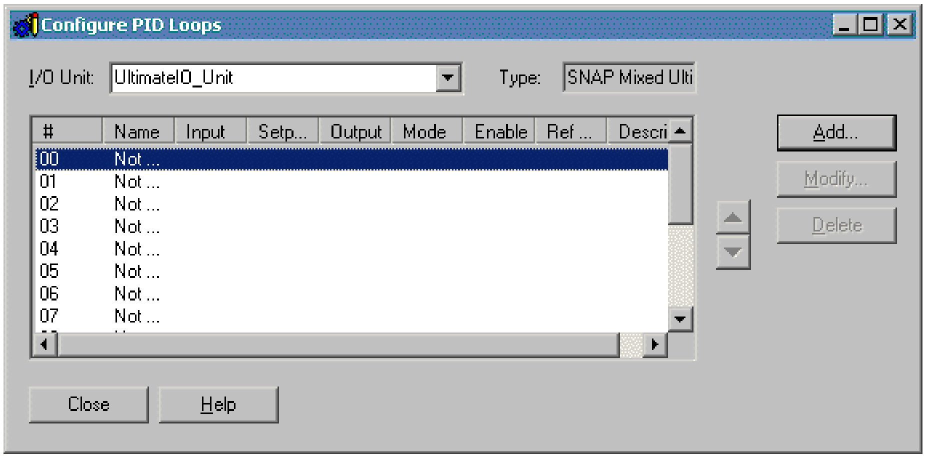
The Configure I/O Units dialog box opens.



Configure I/O Units

Name	Type	Port	Address	Watchdog	Ref Count	Descri
SimpleIO_Unit	SNAP Mi...	Ethernet ...	10.192.5...	Disabled	0	
UltimateIO_Unit	SNAP Mi...	Ethernet ...	10.0.4.1	Disabled	11	

2. Select the I/O unit the PID will be on, and click PID Loops.



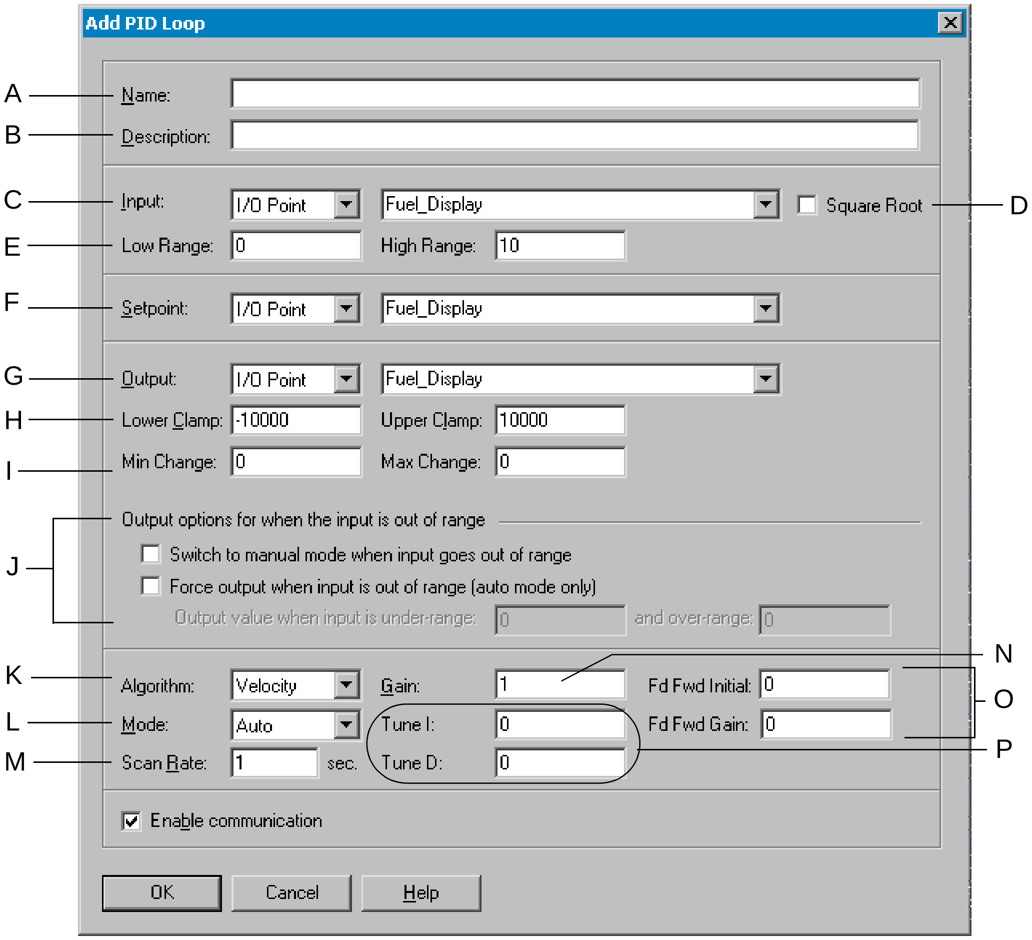
Configure PID Loops

I/O Unit: UltimateIO_Unit

Type: SNAP Mixed Ulti

#	Name	Input	Setp...	Output	Mode	Enable	Ref ...	Descri
00	Not ...							
01	Not ...							
02	Not ...							
03	Not ...							
04	Not ...							
05	Not ...							
06	Not ...							
07	Not ...							

3. Double-click the lowest unused number.



Name:
Description:
Input: I/O Point | Fuel_Display | □ Square Root
Low Range: 0 | High Range: 10
Setpoint: I/O Point | Fuel_Display
Output: I/O Point | Fuel_Display
Lower Clamp: -10000 | Upper Clamp: 10000
Min Change: 0 | Max Change: 0

Output options for when the input is out of range

□ Switch to manual mode when input goes out of range

□ Force output when input is out of range (auto mode only)

Output value when input is under-range: 0 | and over-range: 0

Algorithm: Velocity | Gain: 1 | Fd Fwd Initial: 0
Mode: Auto | Tune I: 0 | Fd Fwd Gain: 0
Scan Rate: 1 sec. | Tune D: 0

☑ Enable communication

4. Complete the fields as described in "Add PID Loop Dialog Box" below.

5. Click OK.

   The new PID appears in the Configure PID Loops dialog box.

6. When you have finished configuring PIDs, click Close.

   PIDs appear in the Strategy Tree under the I/O unit.

Add PID Loop Dialog Box

(A) Name Type a unique, descriptive name for the PID. The name must start with a letter and may contain letters, numbers, and underscores (spaces are converted to underscores).

(B) Description (Optional) Enter a description of the PID.

(C) Input Select the type of input: I/O Point, Host, or PID Output.

- If the PID's process variable comes from an I/O point on the same unit, select I/O Point. Choose the point from the dropdown list or type a point name to configure a new point.

- If the PID's process variable comes from the ioControl strategy, select Host. Enter an initial value for the input.
- If the PID's process variable is the output of another PID on this brain (a cascading control loop), select PID Output. Choose the PID from the dropdown list.

(D) Square Root (Optional) If you chose I/O Point or PID for step C, check this box if the error should be calculated based on the square root of the process variable (applies to flow control systems where volumetric flow is proportional to the square root of a signal from a flow transducer).

(E) Low/High Range Set the valid range of the process variable by entering the low range and the high range. (See Output Options for optional responses to out-of-range input.)

(F) Setpoint Choose the source for the setpoint: I/O Point, Host, or PID Output.

- To control the setpoint using a device (on the same brain) such as a potentiometer, select I/O Point; choose an I/O point from the dropdown list or type a new point name.
- To control setpoint using ioControl or ioDisplay, select Host and enter an initial value.
- If another PID loop will control the setpoint, select PID Output and choose the PID from the dropdown list.

(G) Output Choose the destination for the PID output: I/O Point or Host. (To use the output for controlling the setpoint or input of another PID, choose Host.)

(H) Lower Clamp/Upper Clamp Enter upper and lower clamp values to prevent the output from exceeding a desirable range. These values should equal the range of the output point, if used. Or choose values to make sure that the output device doesn't shut off (for example, keeping a circulation pump running regardless of the PID output) or that the output never reaches a destructively high setting (for example, keeping a motor below maximum).

(I) Min Change/Max Change (Optional) Enter minimum and maximum change values. The output won't respond until the minimum change is reached (for example, you may not want a heater to turn on to correct a 1 degree error). Maximum change prevents too drastic a change in output (for example, you could limit the increase in a pump's output to prevent pipe breakage). The default for both minimum and maximum is zero, which disables the feature.

(J) Output Options Choose how the PID should respond if the input goes out of range. If no boxes are checked, the PID will freeze output at the current value. To have ioControl logic or an operator respond, check Switch to manual mode. To force the output to a specific value, check that option and type the output values.

NOTE: If both boxes are checked (forced output and manual mode), the output will be forced and the PID put into manual mode; but if the PID is already in manual mode, the output will not be forced. (You can use the command Get PID Status Flags to determine current settings.)

(K) Algorithm Choose algorithm: Velocity, ISA, Parallel, Interacting. For details on algorithms, see "Algorithm Choices (PID—Ethernet)" on page 10-60.

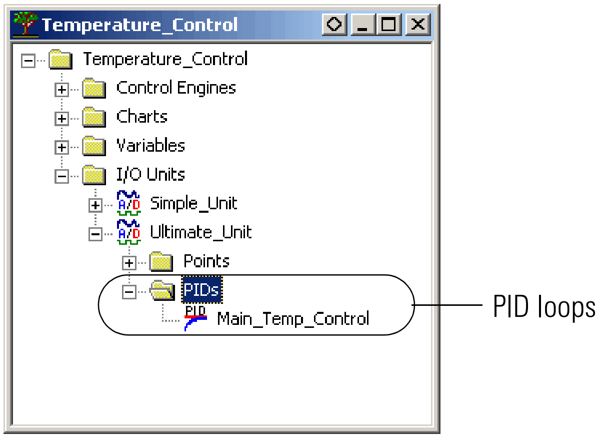
(L) Mode Choose Mode. Auto activates the PID. Manual requires that ioControl logic or an operator control the PID output.

(M) Scan Rate Enter a scan rate to determine how often the input is scanned and the controller output is calculated. Minimum value is 0.001 (1 ms). Scan time should be greater than system lag (the time it takes for the controller output to have a measurable effect on the system). Also consider other PIDs and tasks on the brain competing for processing power.

(N) Gain Type a positive or negative value for Gain. Heating systems usually require a negative value and cooling systems a positive value. NOTE: Gain is usually refined during the tuning process.

(O) Fd Fwd Initial/Fd Fwd Gain (Optional) Enter Feed forward Initial and Feed forward gain values if you need to offset the controller output in your application. These values are constants that are multiplied and added to the controller output; often they are not used in PIDs.

(P) Tune I/Tune D (Optional) Type Integral and Derivative settings if you know the desirable settings. However, Integral and Derivative are not essential to basic configuration and are better determined in the tuning process.

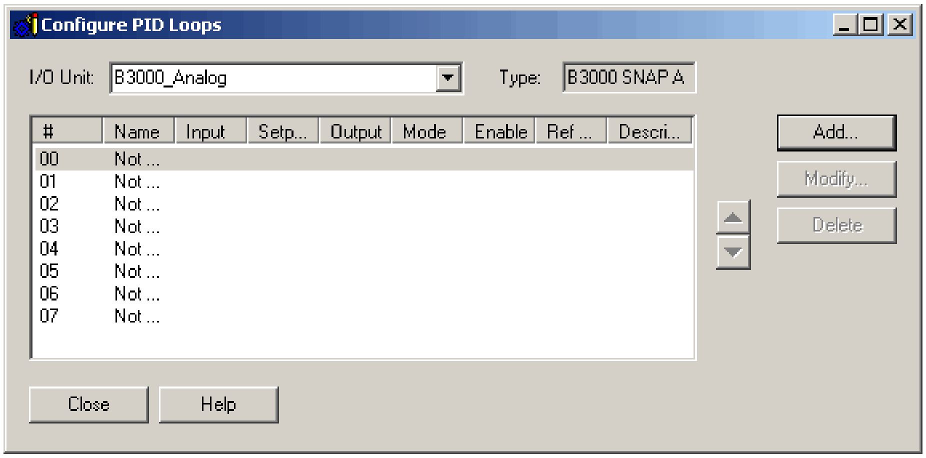


Adding a PID Loop (mistic)

Mistic PID loops can be configured only on B3000 brains and G4A8R bricks. (For PID loops on Ethernet-based I/O units, see page 6-30.)

1. To configure a mistic PID loop, make sure you have the strategy open in Configure mode.
2. On the Strategy Tree, double-click the I/O Units folder (not the individual unit's icon).
3. In the Configure I/O Units dialog box, click the PID Loops button.

The Configure PID Loops dialog box appears, listing all PID loops on the I/O unit.



Configure PID Loops

I/O Unit: B3000_Analog

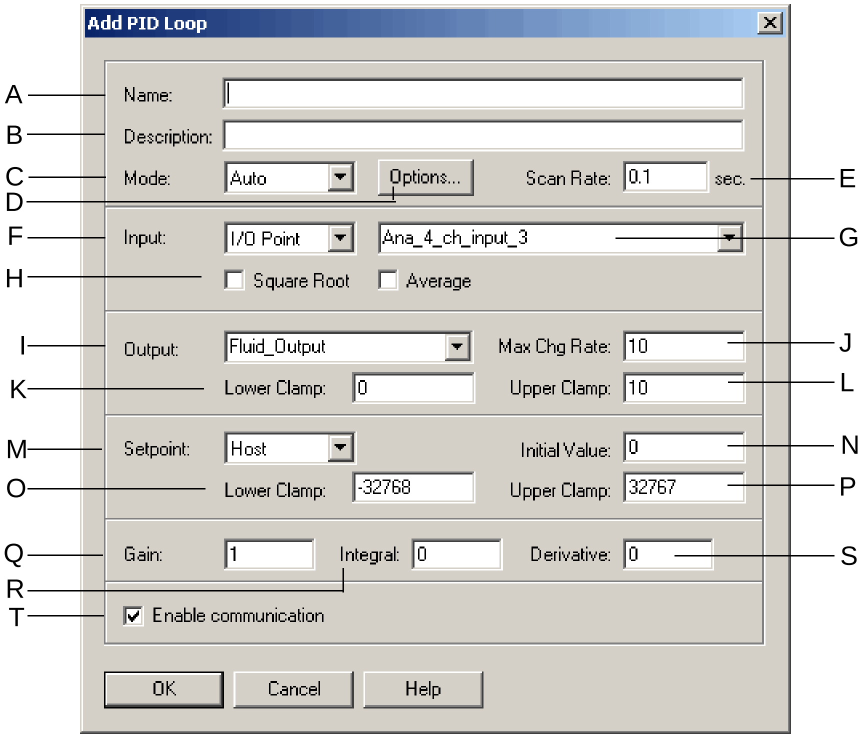
Type: B3000 SNAP A

#	Name	Input	Setp...	Output	Mode	Enable	Ref ...	Descri...
00	Not ...							
01	Not ...							
02	Not ...							
03	Not ...							
04	Not ...							
05	Not ...							
06	Not ...							
07	Not ...							

4. If the correct I/O unit does not appear in the I/O Unit field, choose it from the drop-down menu.

5. Double-click an unused line (or highlight it and click the Add button).

   The Add PID Loop dialog box opens.



Add PID Loop

Name:

Description:

Mode: Auto

Scan Rate: 0.1 sec.

Input: I/O Point   Ana_4_ch_input_3

□ Square Root   □ Average

Output: Fluid_Output

Max Chg Rate: 10

Lower Clamp: 0   Upper Clamp: 10

Setpoint: Host

Initial Value: 0

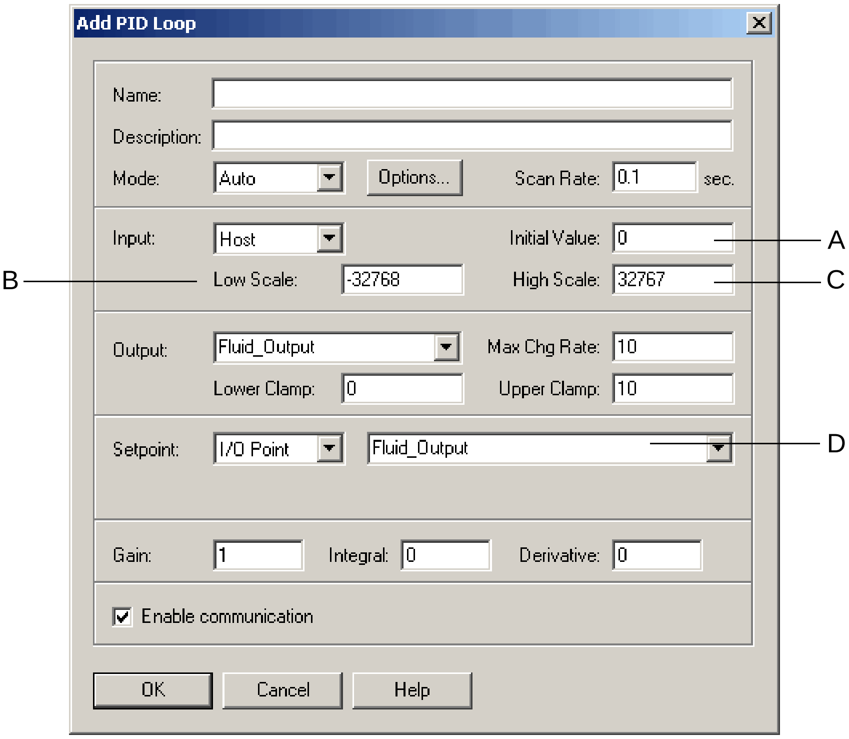
Lower Clamp: -32768   Upper Clamp: 32767

Gain: 1   Integral: 0   Derivative: 0

☑ Enable communication

6. Complete the fields as described in "Add PID Loop Dialog Box (mistic)" below.

7. If you chose Host as the Input source and I/O Point as the Setpoint source, complete the following fields:



Name:

Description:

Mode: Auto

Scan Rate: 0.1 sec.

Input: Host

Initial Value: 0

Low Scale: -32768

High Scale: 32767

Output: Fluid_Output

Max Chg Rate: 10

Lower Clamp: 0

Upper Clamp: 10

Setpoint: I/O Point Fluid_Output

Gain: 1

Integral: 0

Derivative: 0

☑ Enable communication

(A) Initial Value (If Host is selected as Input source) Specify the initial value for the input for the PID calculation. This value must be between the low-scale and high-scale values (B and C). By default, this value is set to zero.

(B) Low Scale (If Host is selected as Input source) Specify the lowest possible input value. The default is -32,768.

(C) High Scale (If Host is selected as Input source) Specify the highest possible input value. The default is 32,767.

(D) I/O point (If I/O Point is selected as Setpoint source) Select from the drop-down list the I/O point providing the setpoint value. If the point doesn't exist, enter a new name and add it.

Add PID Loop Dialog Box (mistic)

(A) Name Enter a name for the PID. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

(B) Description (Optional) Enter a description of the PID.

(C) Mode Select whether the initial mode of the PID is automatic or manual. In Automatic mode, the PID is automatically calculated. In Manual mode, no calculation is made.

(D) Options To set advanced options, click Options and see "Setting PID Loop Control Options (mistic PIDs)" on page 6-38.

(E) Scan Rate Enter a time (in seconds) representing the interval between PID loop calculations. The smaller the number, the more often the PID will be calculated. The default of 0.1 specifies a PID calculation 10 times per second.

(F) Input Select whether the input for the PID calculation will be read from an I/O point or a host device (the controller). Your choice determines the other data required in the Input section of the dialog box. If you choose Host, see the figure in step 7 for additional fields to complete.

(G) Input drop-down list (If I/O Point is selected at F) Select from the drop-down list the I/O point providing the input value. If the point doesn't exist, enter a new name and add the point.

(H) Square Root (If I/O Point is selected at F) Enable or disable square root extraction and averaging of the input value prior to the PID calculation. These options are disabled by default.

(I) Output Specify the output to be driven by the PID by selecting an I/O point from the drop-down list. If it doesn't exist, enter a new I/O point name and add the point.

(J) Maximum Change Rate Specify the maximum absolute difference allowed for the output value as the result of one PID calculation. For example, a maximum change rate of 10 specifies that even if a PID calculation suggests an output change of 20 units, the maximum change allowed during the loop will be 10 units.

You can use a maximum change rate to avoid sudden, dramatic increases or decreases in the output value. Note that the maximum change rate must be between one percent and 100 percent of the range of the output itself, which is the difference between the output's high-scale value (L) and low-scale value (K). The value representing this full range will appear by default.

(K) Lower Clamp, Output Specify the minimum value allowable for the output. This value cannot be less than the zero-scale value of the output module.

(L) Upper Clamp, Output Specify the maximum value allowable for the output. This value cannot be greater than the full-scale value of the output module.

(M) Setpoint Select whether the setpoint for the PID calculation will be read from an I/O point or a host device (the controller). The setpoint is the value to which the input will be driven. Your choice determines the other data required in the Setpoint section of the dialog box. If you choose I/O Point, see the figure in step 7 for additional fields to complete.

(N) Initial Value (If Host is selected at M) Specify the initial value for the setpoint. This value must be between the lower-clamp and upper-clamp values (O and P). The default is zero.

(O) Lower Clamp, Setpoint (If Host is selected at M) Specify the lowest possible setpoint value. Default: -32,768.

(P) Upper Clamp, Setpoint (If Host is selected at M) Specify the highest possible setpoint value. Default: 32,767.

(Q) Gain Specify the gain term (P) to be used in the PID calculation. This value can range between -32,768 and 32,767 but must not be zero. The default is one.

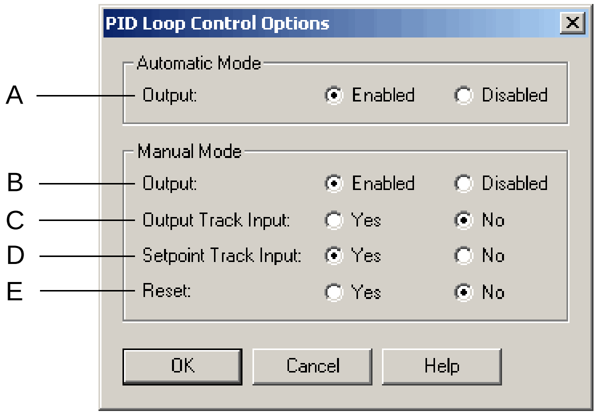
(R) Integral Specify the integral term (I) to be used in the PID calculation. This value can range between zero and 32,767. The default is zero. (Note: The product of the scan rate and the integral term must be less than or equal to 3,932,100.)

(S) Derivitive Specify the derivative term (D) to be used in the PID calculation. This value can range between zero and 32,767. The default is zero.

(T) Enable Communication Select whether you want communication to this PID loop enabled or disabled.

Setting PID Loop Control Options (mistic PIDs)

If you clicked the Options button in the Add PID Loop dialog box in step 6 of "Adding a PID Loop (mistic)" on page 6-34, the PID Loop Control dialog box opens.



PID Loop Control Options

Automatic Mode

Output: ● Enabled ○ Disabled

Manual Mode

Output: ● Enabled ○ Disabled

Output Track Input: ○ Yes ● No

Setpoint Track Input: ● Yes ○ No

Reset: ○ Yes ● No

1. Set the options as described in "PID Loop Control Options Dialog Box" below.

   NOTE: Since you can switch between automatic and manual modes through the View PID Loop dialog box in Debug mode, you may want to configure options for both modes.

2. When you have set the options, click OK to return to the Add PID Loop dialog box.

PID Loop Control Options Dialog Box

(A) Automatic Mode If you want the PID calculation transferred to the output, click Enabled (the default). If you do not want the PID calculation transferred to the output, click Disabled. In either case, the PID will continue calculating and will not be reset.

(B) Manual Mode/Output If you want the PID calculation transferred to the output, click Enabled (the default). If you do not want the PID calculation transferred to the output, click Disabled.

(C) Manual Mode/Output Track Input If you want the output to assume the input value, click Yes. You can use this option to create a signal converter (for example, 4–20 mA input to 0–10 V output), since the output is proportional to the input.

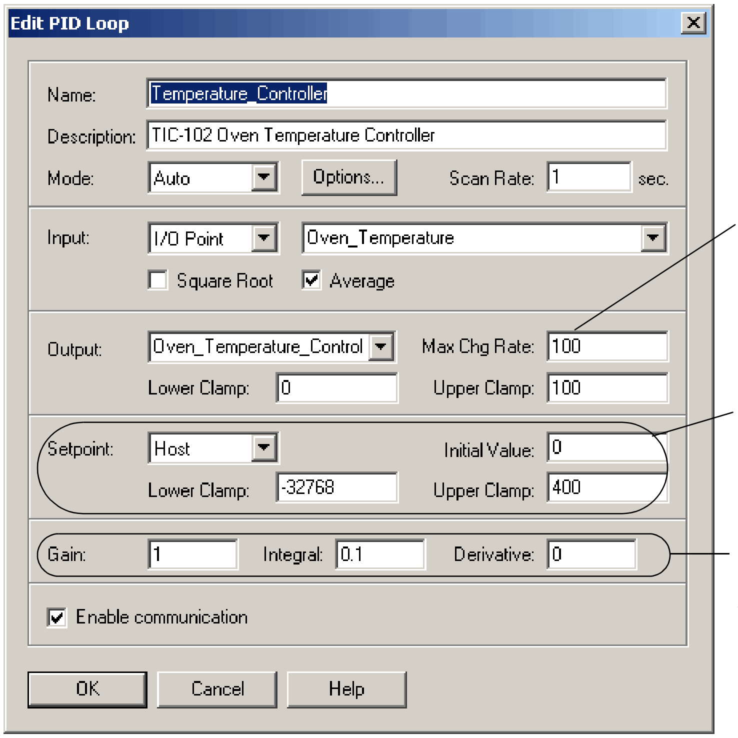
(D) Manual Mode/Setpoint Track Input If you want the setpoint to be set to the input value, click Yes. You can use this option to smooth the transfer when returning to automatic mode.

(E) Manual Mode/Reset If you want to force the PID loop to reset when entering manual mode, click Yes. This option stops all calculations, sets all process errors to zero, and resets the scan rate timer, leaving the output unchanged. If you want the PID calculation to continue, click No. (When you enter automatic mode, however, you cannot force a reset.)

Mistic PID Loop Configuration Example

Here's an example of a completed mistic PID configuration for a temperature controller. The PID modifies an output automatically every second.

The input is an I/O point called Oven_Temperature. Its value is averaged before being applied to the PID calculation. The output being driven is an I/O point called Oven_Temperature_Control, which must remain between zero and 100 units.



Name: Temperature_Controller

Description: TIC-102 Oven Temperature Controller

Mode: Auto

Scan Rate: 1 sec.

Input: I/O Point — Oven_Temperature

☐ Square Root ☑ Average

Output: Oven_Temperature_Control

Max Chg Rate: 100

Lower Clamp: 0

Upper Clamp: 100

Setpoint: Host

Initial Value: 0

Lower Clamp: -32768

Upper Clamp: 400

Gain: 1

Integral: 0.1

Derivative: 0

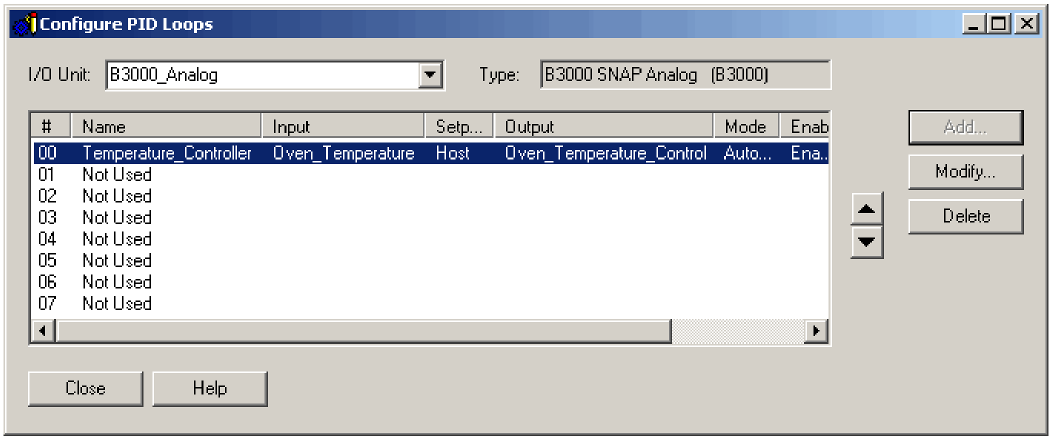
☑ Enable communication

The maximum change rate has been set to the full range (100), which means we are allowing the output to change as much as it needs to during each cycle.

The setpoint is read from the host device (the controller) and is initially set to zero. It is clamped at 400 at the upper end.

For the PID calculation, the gain term is one, the integral term is 0.1, and the derivative term is zero.

The sample PID would appear in the Configure PID Loops dialog box like this:



I/O Unit: B3000_Analog

Type: B3000 SNAP Analog (B3000)

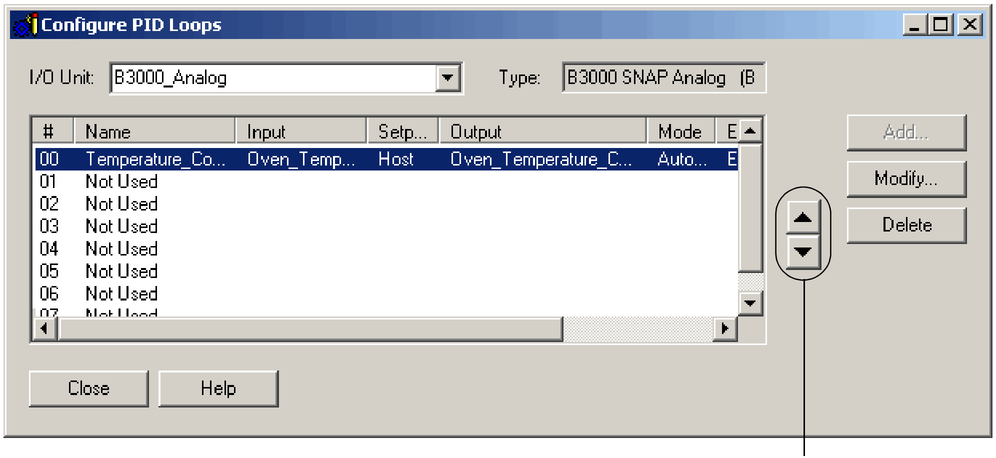
#	Name	Input	Setp...	Output	Mode	Enab
00	Temperature_Controller	Oven_Temperature	Host	Oven_Temperature_Control	Auto...	Ena..
01	Not Used					
02	Not Used					
03	Not Used					
04	Not Used					
05	Not Used					
06	Not Used					
07	Not Used					

Changing a PID Loop (Ethernet or mistic)

You can change the PID loop's configuration and its position in the I/O unit.

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, expand the I/O Units folder until you see the PIDs folder for the I/O unit you want. Double-click the PIDs folder.

   The Configure PID Loops dialog box opens, listing all configured PID loops. Remember that the number of PID loops available depends on the I/O unit.



Configure PID Loops

I/O Unit: B3000_Analog

Type: B3000 SNAP Analog (B

#	Name	Input	Setp...	Output	Mode	E
00	Temperature_Co...	Oven_Temp...	Host	Oven_Temperature_C...	Auto...	E
01	Not Used					
02	Not Used					
03	Not Used					
04	Not Used					
05	Not Used					
06	Not Used					
07	Not Used					

Up- and down- arrows

   PID loops are scanned by the I/O unit in the order that they appear in this list.

2. To move the PID loop to a different position on the I/O unit, use the up- and down-arrows in the dialog box.

3. To change the PID loop's configuration, double-click its name to open the Edit PID Loop dialog box. Change the fields as necessary.

   For help in completing the fields, see "Adding a PID Loop (Ethernet)" on page 6-30 or "Adding a PID Loop (mistic)" on page 6-34.

Deleting a PID Loop (Ethernet or mistic)

Only PID loops that have a reference count of zero can be deleted. Be careful when deleting PID loops; you cannot undo a deletion.

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, expand the I/O units folder until you see the PID loop you want to delete.

2. Right-click the name of the PID loop and choose Delete from the pop-up menu.

   The PID loop is deleted.

You can also delete a PID loop in the Configure PID Loops dialog box by highlighting it and clicking Delete.

Configuring Event/Reactions

Event/reactions apply to **serial-based mistic I/O units only**. If you are moving from OptoControl to ioControl, note that **the interrupt line is not currently supported** in ioControl.

NOTE: Similar events and reactions can be configured on a SNAP Ethernet-based I/O unit using ioManager, but they can interfere with ioControl strategy logic unless you are very careful. For more information, see the ioManager User's Guide (Opto 22 form #1440)

Event/reactions do exactly what their name suggests: they make something happen in response to an event. Their advantage is that they simplify and speed the control engine's job by offloading certain control functions from the control engine to the I/O unit.

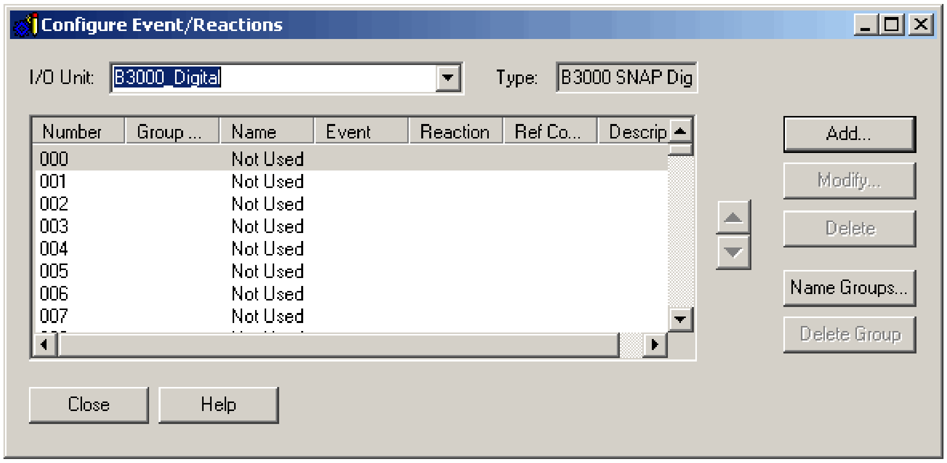
Examples of events are a timeout, an input or output reaching a value, or a special digital feature (such as frequency or an on-time totalizer) reaching a value. Examples of reactions include starting an on- or off-pulse, reading and holding a value, or activating or deactivating a mistic PID loop.

Event/reactions can be configured on mistic digital and analog multifunction I/O units only. You can configure up to 256 event/reactions on a single I/O unit. For more information on using event/reactions, see "Event Reaction Commands" on page 10-22.

1. To configure an event/reaction, make sure the strategy is open in Configure mode. On the Strategy Tree, double-click the I/O Units folder (not the individual unit's icon).

2. In the Configure I/O Units dialog box, highlight the multifunction mistic I/O unit and click the Event/Reactions button.

   The Configure Event/Reaction dialog box appears, showing all event/reactions contained on the I/O unit (in this example, none).



Configure Event/Reactions

I/O Unit: B3000_Digital

Type: B3000 SNAP Dig

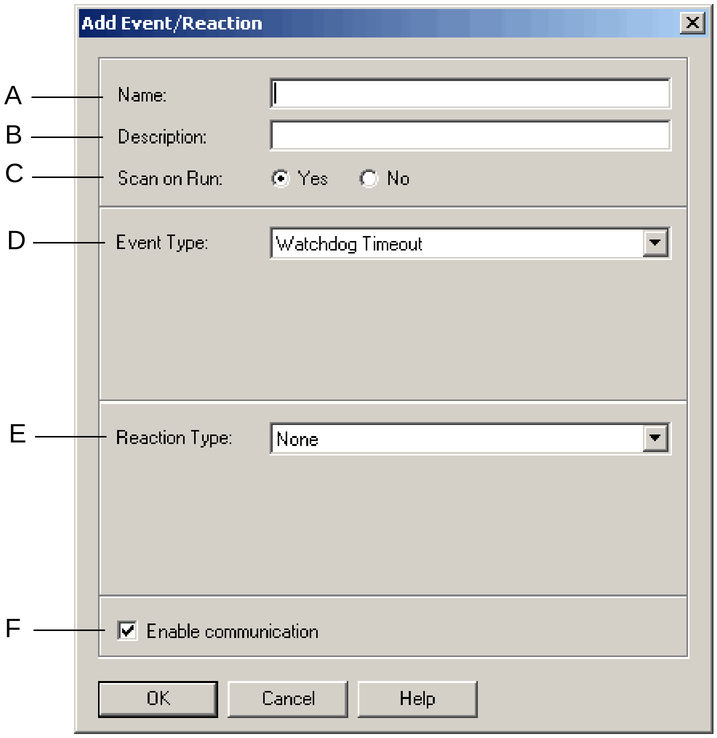
Number	Group ...	Name	Event	Reaction	Ref Co...	Descrip
000		Not Used				
001		Not Used				
002		Not Used				
003		Not Used				
004		Not Used				
005		Not Used				
006		Not Used				
007		Not Used				

3. If the correct I/O unit is not shown, select the unit from the drop-down list.

   Since only multifunction units support event/reactions, they are the only units listed.

4. Highlight an unused line and click Add, or double-click an unused line.

   The Add Event/Reaction dialog box appears.



Add Event/Reaction

Name:

Description:

Scan on Run: ● Yes ○ No

Event Type: Watchdog Timeout

Reaction Type: None

☑ Enable communication

This figure shows the fields in the dialog box when you first open it.

Depending on the type of event or reaction you select, other fields may appear.

5. Complete the fields as described in "Add Event/Reaction Dialog Box" below.

6. Click OK.

   The event/reaction appears in the Configure Event/Reaction dialog box.

Add Event/Reaction Dialog Box

(A) Name Enter a name for the event/reaction. The name must start with a letter and may contain letters, numbers, and underscores (spaces are converted to underscores).

(B) Description (Optional) Enter a description of the event/reaction.

(C) Scan on Run If you want the I/O unit to begin scanning for the event automatically as soon as the strategy is run, click Yes. If you want scanning to wait until it is started by a command in the strategy, click No.

(D) Event Type From the drop-down list, select an event to scan for. Complete additional fields that appear as described in the following tables.

For digital I/O units:

For this type of event	Enter this information
Watchdog Timeout	No information required.
Counter >= Value Quadrature >= Value Totalize On >= Value Totalize Off >= Value On-Pulse >= Value Off-Pulse >= Value Period >= Value Frequency >= Value Quadrature <= Value Frequency <= Value Counter <= Value	Specify the I/O point to be monitored and the value to compare the I/O point against. Select the I/O point from the drop-down list or specify a new name for the point and configure it.
MOMO Match	See "Adding a MOMO Event or Reaction (mistic I/O Units Only)" on page 6-46.

For analog I/O units:

For this type of event	Enter this information
Watchdog Timeout	No information required.
Analog Input >= Value Analog Input <= Value Analog Output >= Value Analog Output <= Value	Specify the I/O point to be monitored, the value to compare the I/O point against, and the type of comparison value (current, average, maximum, minimum, or total). Select the I/O point from the drop-down list or specify a new name for the point and configure it.

(E) Reaction Type From the drop-down list, select the reaction to take in response to the event. Complete additional fields that appear as described in the following tables.

For digital I/O units:

For this type of reaction	Enter this information
None (no reaction)	None
Enable Scan for Event Disable Scan for Event	Select the event from the drop-down list.
Enable Scan for E/R Group Disable Scan for E/R Group	Select the group from the drop-down list.
Disable Scan for All Events	None
Set MOMO Outputs	See "Adding a MOMO Event or Reaction (mistic I/O Units Only)" on page 6-46.

For this type of reaction	Enter this information
Start On-Pulse Start Off-Pulse	Specify the I/O point to be pulsed and the length of the pulse in seconds. Select the I/O point from a drop-down list or specify a new name for the point and configure it.
Start Counter Stop Counter Start Quadrature Counter Stop Quadrature Counter Clear Counter Clear Quadrature Counter Clear On-Pulse Clear Off-Pulse Clear Period Clear Totalize On Clear Totalize Off Read and Hold Counter Value Read and Hold Quadrature Value Read and Hold Totalize On Value Read and Hold Totalize Off Value Read and Hold On-Pulse Value Read and Hold Off-Pulse Value Read and Hold Period Value Read and Hold Frequency Value	Specify the I/O point to be affected. Select the I/O point from a drop-down list or specify a new name for the point and configure it.

For analog I/O units:

For this type of reaction	Enter this information
None (no reaction)	None
Enable Scan for Event Disable Scan for Event	Select the event from the drop-down list.
Enable Scan for E/R Group Disable Scan for E/R Group	Select the group from the drop-down list.
Disable Scan for All Events	None
Read and Hold Analog Input Data Read and Hold Analog Output Data	Specify the I/O point and type of data (current, average, maximum, minimum or total) to be read. Select the I/O point from a drop-down list or specify a new name for the point and configure it.
Activate PID Loop Deactivate PID Loop	Specify the PID loop to be affected. Select the PID from a drop-down list or specify a new name for the PID and configure it.
Set PID Loop Setpoint	Specify the PID loop to be affected and the setpoint value. Select the PID from a drop-down list or specify a new name for the PID and configure it.

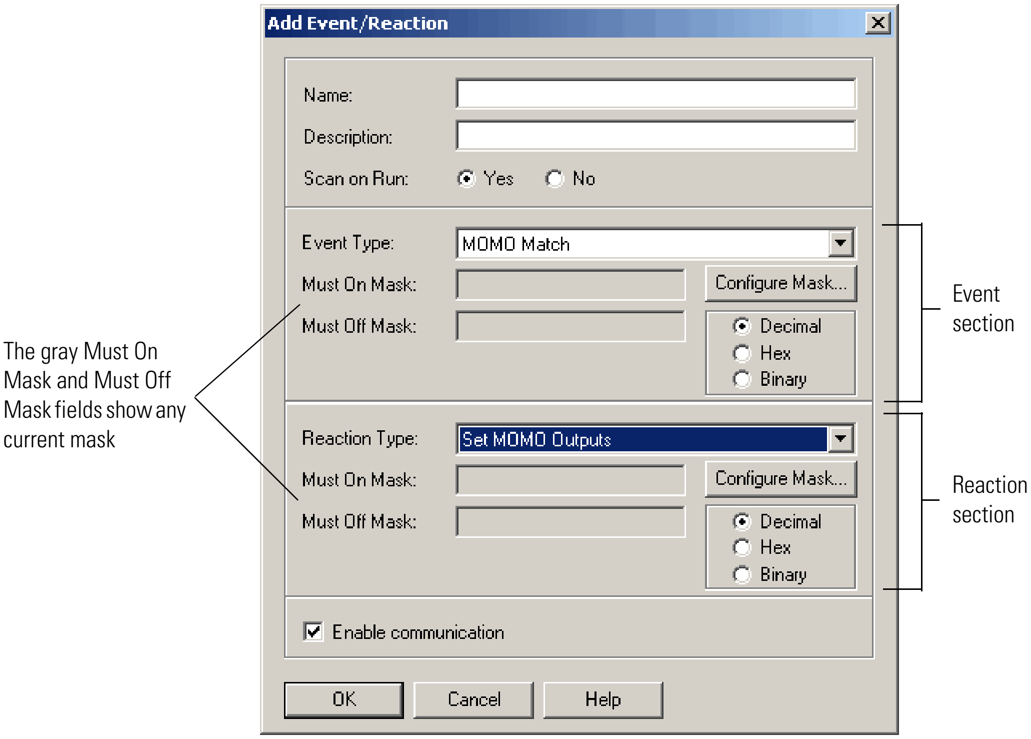
For this type of reaction	Enter this information
Set Analog Output	Specify the I/O point and the value to set it to. Select the I/O point from a drop-down list or specify a new name for the point and configure it.
Ramp Analog Output to Setpoint	Specify the I/O point, the ramping speed in units per second, and the end point value to ramp to. Select the I/O point from a drop-down list or specify a new name for the point and configure it.

(F) Enable Communication To enable communication to this event/reaction, check to enable communication to this event/reaction, or uncheck to disable communication.

Adding a MOMO Event or Reaction (mistic I/O Units Only)

On a digital multifunction I/O unit, a special type of event or reaction you can configure is called MOMO (must-on, must-off). A MOMO Match event monitors several inputs and/or outputs on an I/O unit for a match to a specific pattern. A Set MOMO Outputs reaction defines a set of values for outputs on an I/O unit. You can use just a MOMO event, just a MOMO reaction, or both.

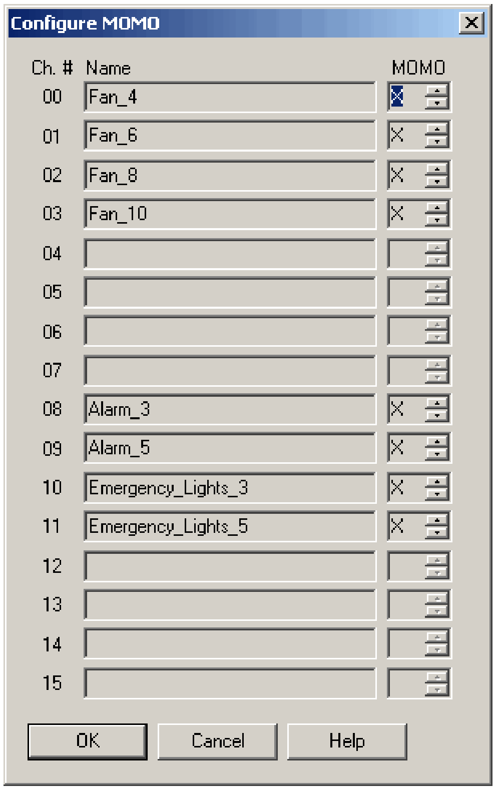
The following figure shows the additional fields and buttons that appear if you select the MOMO Match event or the Set MOMO Outputs reaction:



The gray Must On Mask and Must Off Mask fields show any current mask

1. To set up or change the must-on pattern or must-off pattern (called the mask), click Configure Mask in the Event or Reaction sections.

The Configure MOMO dialog box appears, listing all I/O points you can set in the mask.



Ch. #	Name	MOMO
00	Fan_4	X
01	Fan_6	X
02	Fan_8	X
03	Fan_10	X
04		
05		
06		
07		
08	Alarm_3	X
09	Alarm_5	X
10	Emergency_Lights_3	X
11	Emergency_Lights_5	X
12		
13		
14		
15		

2. For each I/O point, click the scroll arrows to indicate that the point is On, Off, or X (ignored). When you have finished the mask, click OK.

You return to the Add Event/Reaction dialog box, and the numerical equivalent of the mask you have set appears in the Must On Mask and Must Off Mask fields.

3. In the Event and Reaction sections, choose whether to display the mask in decimal, hexadecimal, or binary form. The default is decimal.

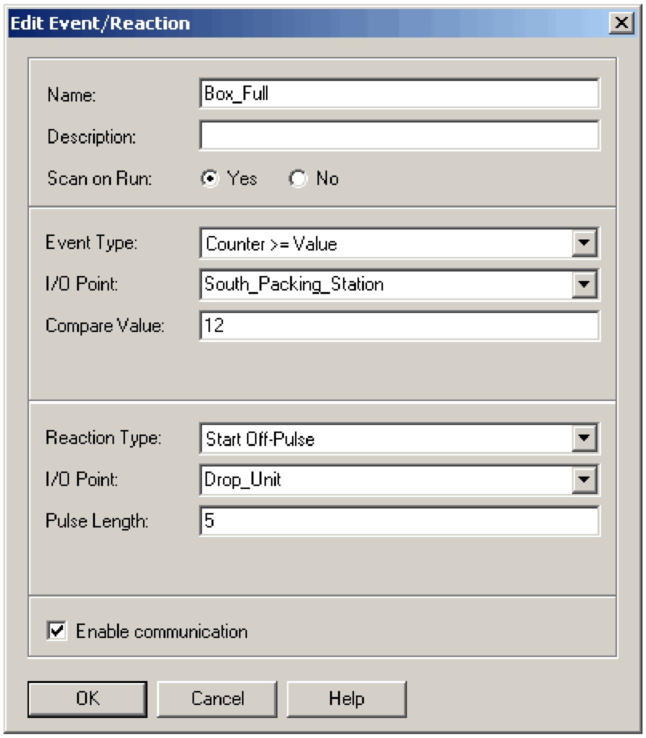
4. When the event/reaction is configured, click OK.

The new event/reaction appears in the Configure Event/Reaction dialog box.

Event/Reaction Configuration Example

NOTE: Event/reactions are available on serial mistic I/O units only.

Here's an example of an Add Event/Reaction dialog box for an event/reaction involving a counter:



Name: Box_Full

Description:

Scan on Run: ● Yes ○ No

Event Type: Counter >= Value

I/O Point: South_Packing_Station

Compare Value: 12

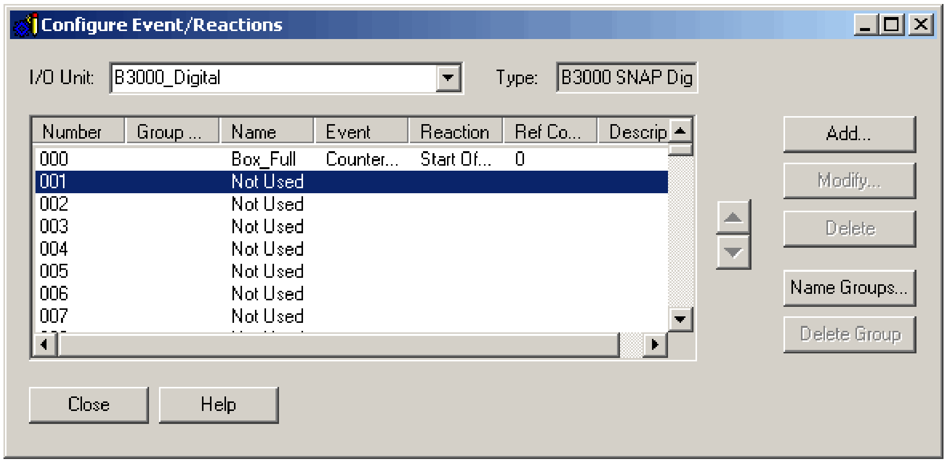
Reaction Type: Start Off-Pulse

I/O Point: Drop_Unit

Pulse Length: 5

☑ Enable communication

This event/reaction would appear in the Configure Event/Reactions dialog box like this:



I/O Unit: B3000_Digital

Type: B3000 SNAP Dig

Number	Group ...	Name	Event	Reaction	Ref Co...	Descrip
000		Box_Full	Counter...	Start Of...	0	
001		Not Used				
002		Not Used				
003		Not Used				
004		Not Used				
005		Not Used				
006		Not Used				
007		Not Used				

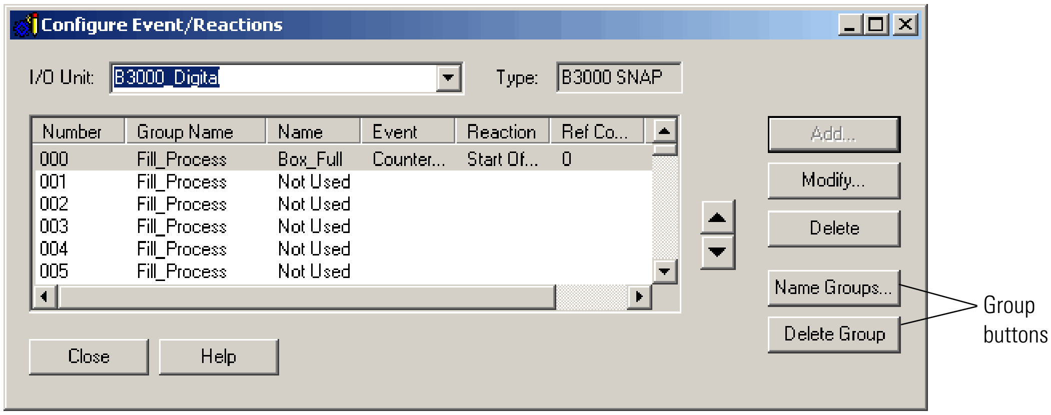
Using Event/Reaction Groups (mistic I/O Units Only)

Since you can configure up to 256 event/reactions on an I/O unit, it's useful to be able to divide them into groups of 16. By grouping related event/reactions, you can also take advantage of commands that start or stop scanning of all event/reactions in a group.

Creating Groups

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, expand the I/O Units folder until you see the E/Rs folder for the I/O unit you want. Double-click the E/Rs folder.

   The Configure Event/Reaction dialog box opens, listing all configured event/reactions.

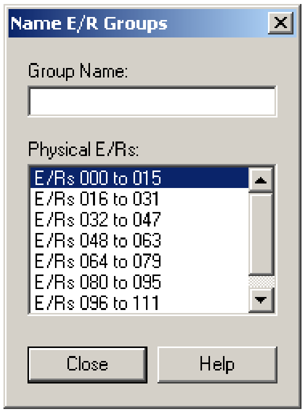


Configure Event/Reactions

I/O Unit: B3000_Digital   Type: B3000 SNAP

Number	Group Name	Name	Event	Reaction	Ref Co...
000	Fill_Process	Box_Full	Counter...	Start Of...	0
001	Fill_Process	Not Used			
002	Fill_Process	Not Used			
003	Fill_Process	Not Used			
004	Fill_Process	Not Used			
005	Fill_Process	Not Used			

2. To create a group, click the Name Groups button.



Name E/R Groups

Group Name:

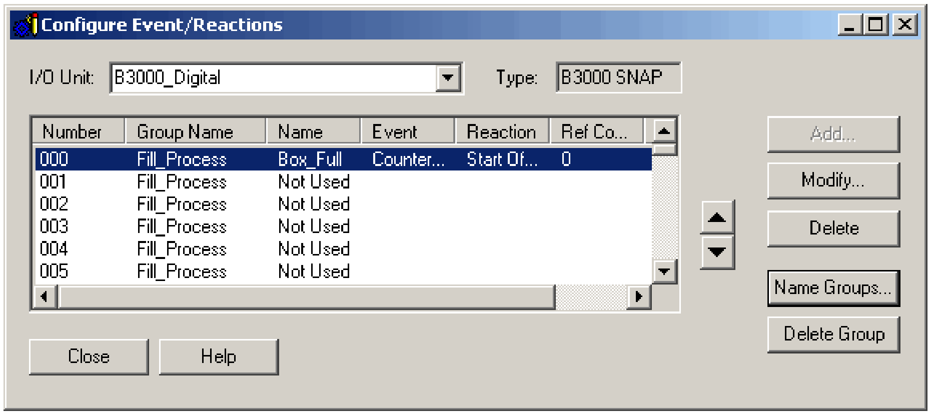
Physical E/Rs:

- E/Rs 000 to 015
- E/Rs 016 to 031
- E/Rs 032 to 047
- E/Rs 048 to 063
- E/Rs 064 to 079
- E/Rs 080 to 095
- E/Rs 096 to 111

3. Highlight a physical group of E/Rs. Click in the Group Name field and type a name.

   Group names must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

4. When you have finished naming groups, click Close. The names appear in the Group Name column of the Configure Event/Reaction dialog box.



I/O Unit: B3000_Digital

Type: B3000 SNAP

Number	Group Name	Name	Event	Reaction	Ref Co...
000	Fill_Process	Box_Full	Counter...	Start Of...	0
001	Fill_Process	Not Used			
002	Fill_Process	Not Used			
003	Fill_Process	Not Used			
004	Fill_Process	Not Used			
005	Fill_Process	Not Used			

Deleting Groups

When you delete a group, you're deleting only the group name, not the event/reactions that were assigned to the group.

To delete a group name, in the Configure Event/Reaction dialog box, select any event/reaction in the group and click the Delete Group button.

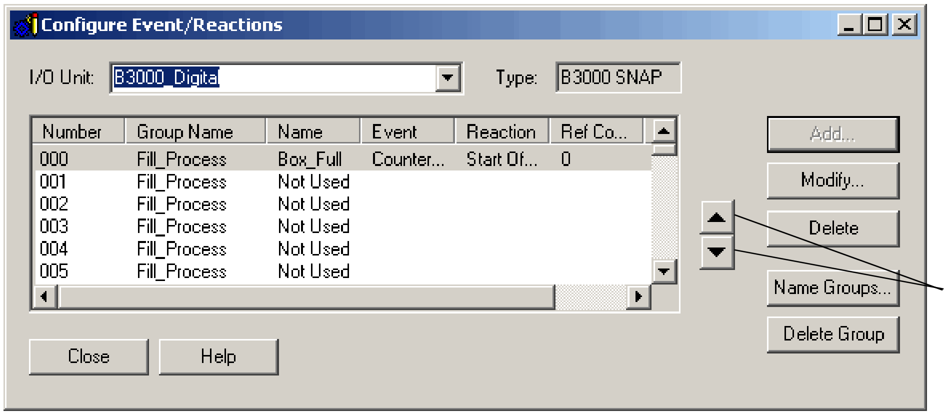
The group name is removed from all 16 event/reactions in the group, and the selected event/reaction appears at the top of the list box.

Changing Configured Event/Reactions (mistic I/O Units Only)

You can change an event/reaction's configuration and its position in the I/O unit.

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, expand the I/O Units folder until you see the E/Rs folder for the I/O unit you want. Double-click the E/Rs folder.

The Configure Event/Reaction dialog box opens, listing all configured event/reactions.



I/O Unit: B3000_Digita

Type: B3000 SNAP

Number	Group Name	Name	Event	Reaction	Ref Co...
000	Fill_Process	Box_Full	Counter...	Start Of...	0
001	Fill_Process	Not Used			
002	Fill_Process	Not Used			
003	Fill_Process	Not Used			
004	Fill_Process	Not Used			
005	Fill_Process	Not Used			

Up- and down- arrows

Event/reactions are scanned by the I/O unit in the order that they appear in this list.

2. To move an event/reaction to a different position on the I/O unit, use the up- and down-arrows in the dialog box.

3. To change an event/reaction's configuration, double-click its name to open the Edit Event/Reaction dialog box. Change the fields as necessary.

   For help in completing the fields, see "Configuring Event/Reactions" on page 6-42.

Deleting Event/Reactions (mistic I/O Units Only)

You can delete only event/reactions with a reference count of zero. Be careful when you delete; you cannot undo a deletion.

1. Make sure the strategy is open and in Configure mode. On the Strategy Tree, expand the I/O units folder until you see the event/reaction you want to delete.

2. Right-click the name of the event/reaction and choose Delete from the pop-up menu.

   The event/reaction is deleted.

   You can also delete an event/reaction in the Configure Event/Reaction dialog box by highlighting it and clicking Delete.

Inspecting I/O in Debug Mode

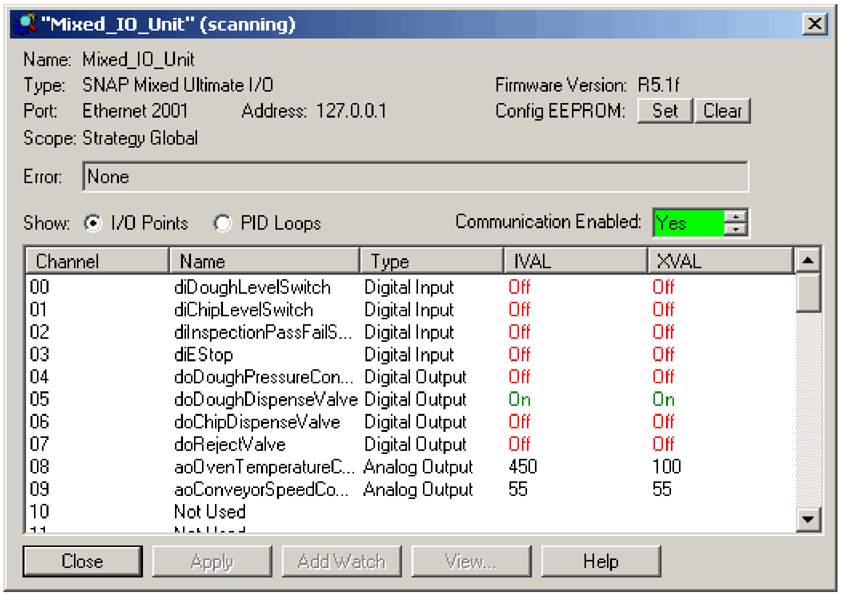
You may want to inspect or change I/O while you are running your Strategy in Debug mode. This section shows how to view information about I/O and make changes while the strategy is running.

To monitor several I/O elements at once in a window you can save with your strategy, see "Using Watch Windows for Monitoring" on page 6-73.

Inspecting I/O Units

1. With the strategy running in Debug mode, double-click an I/O unit in the Strategy Tree.

   The View I/O Unit dialog box appears, showing information about the unit and its points. The title bar shows the name of the I/O unit and whether scanning is occurring.



"Mixed_IO_Unit" (scanning)

Name: Mixed_IO_Unit
Type: SNAP Mixed Ultimate I/O
Port: Ethernet 2001 Address: 127.0.0.1
Scope: Strategy Global
Firmware Version: R5.1f
Config EEPROM: Set Clear

Error: None

Show: ● I/O Points ○ PID Loops
Communication Enabled: Yes

Channel	Name	Type	IVAL	XVAL
00	diDoughLevelSwitch	Digital Input	Off	Off
01	diChipLevelSwitch	Digital Input	Off	Off
02	diInspectionPassFailS...	Digital Input	Off	Off
03	diEStop	Digital Input	Off	Off
04	doDoughPressureCon...	Digital Output	Off	Off
05	doDoughDispenseValve	Digital Output	On	On
06	doChipDispenseValve	Digital Output	Off	Off
07	doRejectValve	Digital Output	Off	Off
08	aoOvenTemperatureC...	Analog Output	450	100
09	aoConveyorSpeedCo...	Analog Output	55	55
10	Not Used			

NOTE: Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

2. To save the current configuration of the I/O unit to its EEPROM (flash memory), click the SET button.

   The following parameters are saved:

Analog	Digital
• I/O module configuration • Initial output settings • Comm link watchdog time • Temperature conversion type • Input offset and gain settings	• I/O module configuration • Comm link watchdog time

   Saving to flash memory this way is the same as saving to flash by other methods, such as using ioManager (see the *ioManager User's Guide*), and is preferable to using the command Write I/O Unit Configuration to EEPROM.

3. To reset saved parameters to their powerup default values, click the Clear button.

4. To change the I/O unit's current status, click an arrow in the Enable field. Then click Apply.

Yes on a green background means enabled; No on a red background means disabled. If you change it, the background turns magenta until you click Apply.

5. To view an individual I/O point, highlight its name in the list.
   - To add an I/O element to a watch window, click Add Watch. See page 6-73.
   - To open an inspection window to change an I/O point, click View. Then see "Inspecting Digital I/O Points" below for the I/O point you are changing (analog or digital).

6. When you have finished inspecting the I/O unit, click Close.

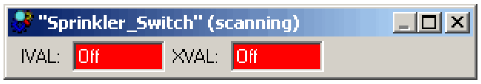
Inspecting Digital I/O Points

You can inspect a Digital Point's data, change its status, or set its Internal Values or External Values in Debug mode. To monitor the point in a watch window, see page 6-73. To change the point, follow these steps.

1. With the strategy running in Debug mode, double-click the I/O point on the Strategy Tree, or double-click the point in the View I/O Unit dialog box.

   The small dialog box that appears shows the IVAL and XVAL.

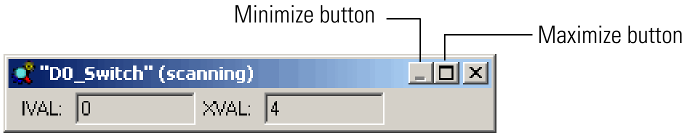
   - The *XVAL*, or external value, is the "real" or hardware value as seen by the I/O unit. This value is external to the control engine.
   - The *IVAL*, or internal value, is a logical or software copy of the XVAL that is in the control engine. The IVAL may or may not be current, since it is updated to match the XVAL only when a strategy in the control engine reads or writes to an I/O point.



"Sprinkler_Switch" (scanning)

IVAL: Off XVAL: Off

   If the digital point is configured with a counter, the counter values appear instead of the point's status.



"DO_Switch" (scanning)

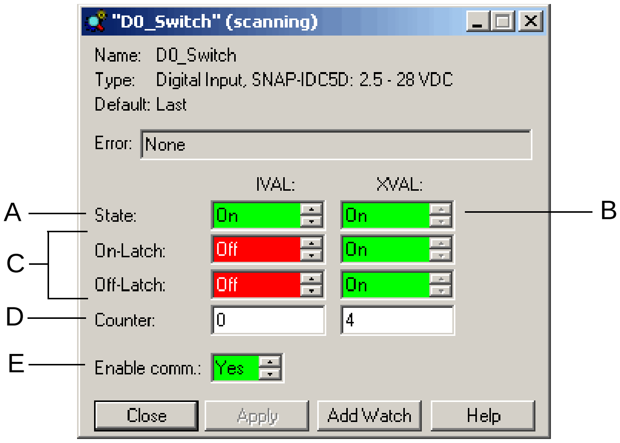
IVAL: 0 XVAL: 4

Minimize button

Maximize button

2. To change the value or to view more information, click the Maximize button.

The title bar shows the name of the digital point and whether scanning is occurring.



"DO_Switch" (scanning)

Name: DO_Switch
Type: Digital Input, SNAP-IDC5D: 2.5 - 28 VDC
Default: Last

Error: None

	IVAL:	XVAL:
State:	On	On
On-Latch:	Off	On
Off-Latch:	Off	On
Counter:	0	4

Enable comm.: Yes

Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

Asterisks in a field indicate an out-of-range value. Dashes in an XVAL field indicate a communication error.

3. Change the fields as necessary:

   (A) State The point's current internal value. Switch between On and Off; then click Apply.

   (B) XVAL The point's current external value. Switch between On and Off; then click Apply.

   (C) On-Latch/Off Latch The state of the point's on and off latches.

   (D) Counter Internal and external feature values if the point has been configured with any special features, such as a counter.

   (E) Enable comm Current point status: Yes on a green background means enabled, No on a red background means disabled. To change the status, click one of the arrows; then click Apply.

4. To add the point to a watch window, click Add Watch and see page 6-73.

Inspecting Analog I/O Points

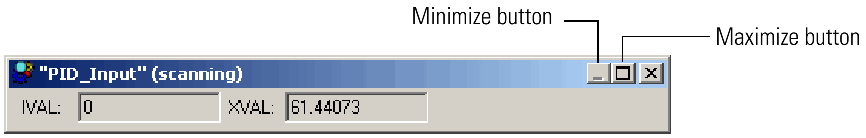
You can review an Analog Point's data, modify its status, or set its Internal Values or External Values in Debug mode. To monitor the point in a watch window, see page 6-73. To change the point, follow these steps.

1. With the strategy running in Debug mode, double-click the I/O point on the Strategy Tree, or double-click the point in the View I/O Unit dialog box.

   The small dialog box that appears shows the IVAL and XVAL, as well as the units.

   - The *XVAL*, or external value, is the "real" or hardware value as seen by the I/O unit. This value is external to the control engine.

- The *IVAL*, or internal value, is a logical or software copy of the XVAL that is in the control engine. The IVAL may or may not be current, since it is updated to match the XVAL only when a strategy in the control engine reads or writes to an I/O point.

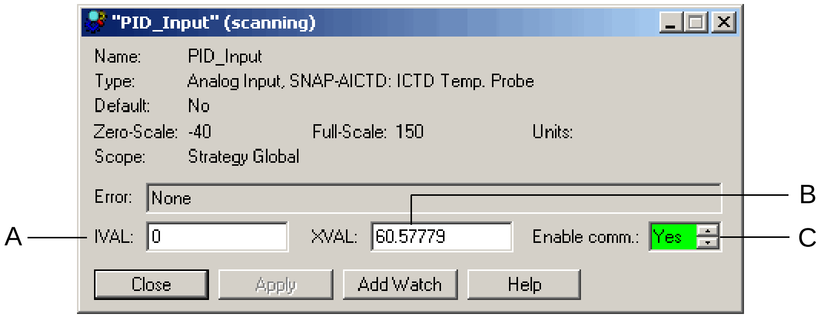


"PID_Input" (scanning)

IVAL: 0 XVAL: 61.44073

2. To change the value or to view more information, click the Maximize button.

   The title bar shows the name of the analog point and whether scanning is occurring.



"PID_Input" (scanning)

Name: PID_Input
Type: Analog Input, SNAP-AICTD: ICTD Temp. Probe
Default: No
Zero-Scale: -40 Full-Scale: 150 Units:
Scope: Strategy Global

Error: None

IVAL: 0 XVAL: 60.57779 Enable comm.: Yes

   Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

   Asterisks in a field indicate an out-of-range value. Dashes in an XVAL field indicate a communication error.

3. Change the fields as necessary:

   (A) IVAL The point's current internal value. You can change it to any value within the valid range of the analog point. For an input, the valid range may exceed the apparent range; that is, you may be able to enter a value lower than the zero-scale value or higher than the full-scale value. For an output however, you cannot enter a value outside of the range defined by the zero-scale and full-scale values. After you change it, click Apply.

   (B) XVAL The point's current external value. You can change it to any value within the valid range of the analog point; then click Apply.

   (C) Enable comm Current point status: Yes on a green background means enabled, No on a red background means disabled. To change the status, click one of the arrows; then click Apply.

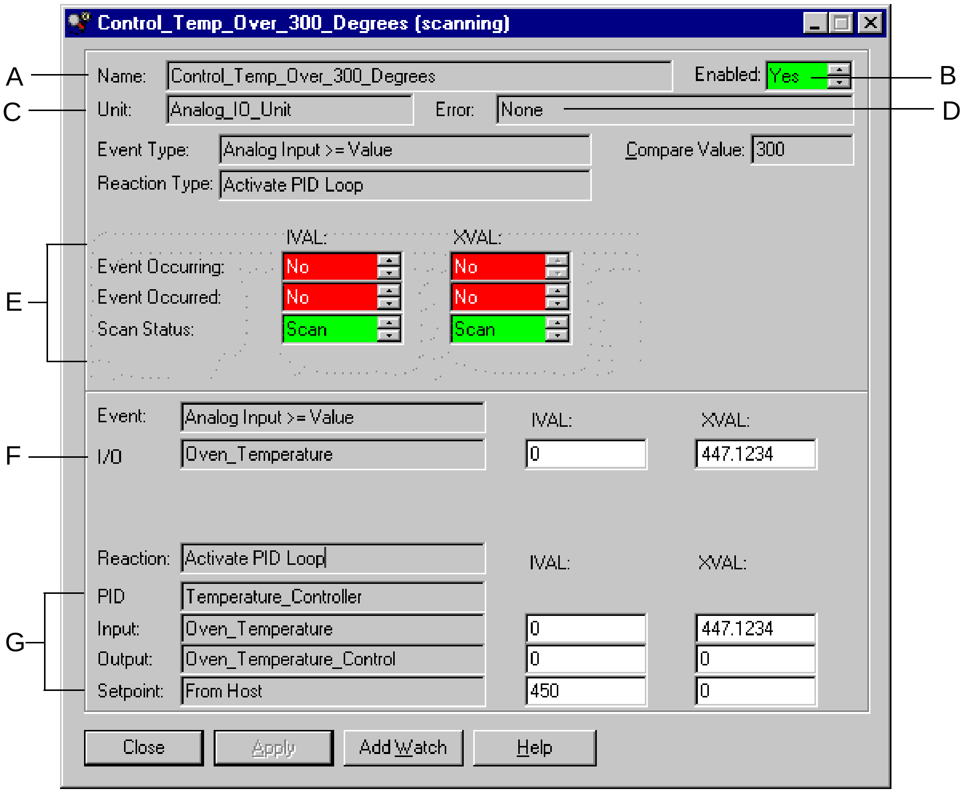
4. To add the point to a watch window, click Add Watch and see page 6-73.

Inspecting Event/Reactions

You can review an event/reaction's current state, modify its status, or set its Internal Values or External Values in Debug mode. To monitor the event/reaction in a watch window, see page 6-73. To change the event/reaction, follow these steps.

1. With the strategy running in Debug mode, double-click the event/reaction on the Strategy Tree, or double-click it in the View I/O Unit dialog box.

   The Event/Reaction dialog box appears, showing the configuration parameters for the event/reaction. The title bar shows the name of the event/reaction and whether scanning is occurring.



Control_Temp_Over_300_Degrees (scanning)

Name: Control_Temp_Over_300_Degrees
Enabled: Yes
Unit: Analog_IO_Unit
Error: None
Event Type: Analog Input >= Value
Compare Value: 300
Reaction Type: Activate PID Loop

	IVAL:	XVAL:
Event Occuring:	No	No
Event Occurred:	No	No
Scan Status:	Scan	Scan

		IVAL:	XVAL:
Event:	Analog Input >= Value		
I/O	Oven_Temperature	0	447.1234
Reaction:	Activate PID Loop		
PID	Temperature_Controller		
Input:	Oven_Temperature	0	447.1234
Output:	Oven_Temperature_Control	0	0
Setpoint:	From Host	450	0

   Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

   Asterisks in a field indicate an out-of-range value. Dashes in an XVAL field indicate a communication error.

2. Change the fields as described in "View Event/Reaction Dialog Box" below.

3. To add the event/reaction to a watch window, click Add Watch and see page 6-73.

View Event/Reaction Dialog Box

(A) Name Name of the event/reaction

(B) Enabled Current status: Yes on a green background means enabled, No on a red background means disabled. To change status, click one of the arrows; then click Apply.

(C) Unit The I/O unit on which the event/reaction is configured

(D) Error Any communication error appears here with a red background.

(E) Event Occurring / Event Occurred / Scan Status Internal and external values for whether the event occurred or is occurring, whether scanning is occurring, and whether an interrupt is to be generated when the event occurs. If the reaction type involves a read-and-hold operation, a Read & Hold Value field also appears in this area, together with its internal and external values. You can change any of the internal or external values. If you do, click Apply.

(F) Event Parameters Parameters of the event, together with their internal and external values. (This example shows an I/O point being monitored for the event. For a MOMO Match event, MOMO data will appear instead. See "MOMO Event/Reactions" on page 6-57.) You can change the internal or external values. If you do, click Apply.

(G) Reaction Parameters Parameters of the reaction, together with their internal and external values. (This example shows the PID to be activated and the sources of its input, output, and setpoint values. We also see the internal and external values of the input, output, and setpoint.) Depending on the reaction type, other parameters that can appear here include an I/O point, an event to be triggered, or MOMO data. You can change any of the internal or external values. If you do, click Apply.

MOMO Event/Reactions

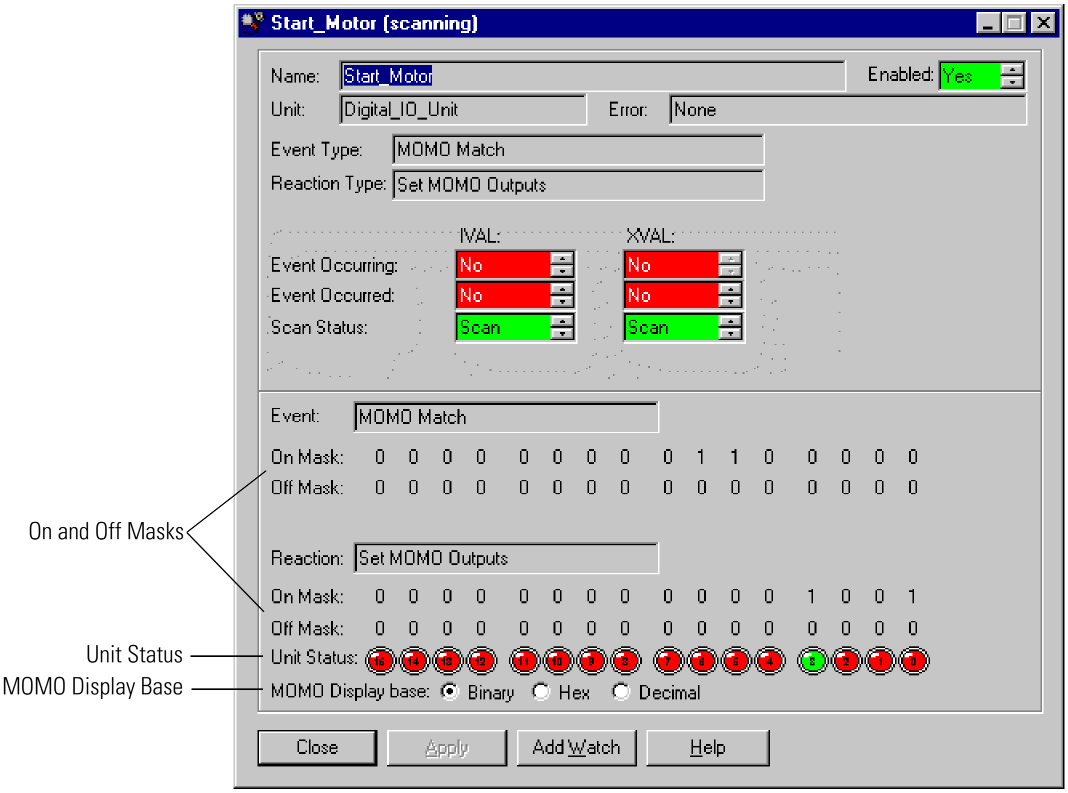
When a MOMO (must-on, must-off) event or reaction is involved, the bottom of the event/reaction dialog box displays the following:

On and Off Masks Shows On and Off Masks for the MOMO event and reaction.

Unit Status Shows LEDs representing the external value of the digital I/O unit on which the event/reaction is configured. Green represents one (on), red represents zero (off), and gray represents no value reported. If there is no MOMO reaction, this set of LEDs appears below the event's on and off masks.

Momo Display Base Shows the display base for the MOMO event or reaction. You can switch among binary, hex, and decimal values.

You cannot change the Internal Values or External Values of the MOMO masks or of the I/O unit status. However, other fields are the same as for other event/reactions.



Start_Motor (scanning)

Name: Start_Motor   Enabled: Yes

Unit: Digital_IO_Unit   Error: None

Event Type: MOMO Match

Reaction Type: Set MOMO Outputs

	IVAL:	XVAL:
Event Occurring:	No	No
Event Occurred:	No	No
Scan Status:	Scan	Scan

Event: MOMO Match

On Mask:	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
Off Mask:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reaction: Set MOMO Outputs

On Mask:	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
Off Mask:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Unit Status:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

MOMO Display base: ● Binary ○ Hex ○ Decimal

The example above shows on and off masks for both the MOMO Match event and the Set MOMO Outputs reaction.

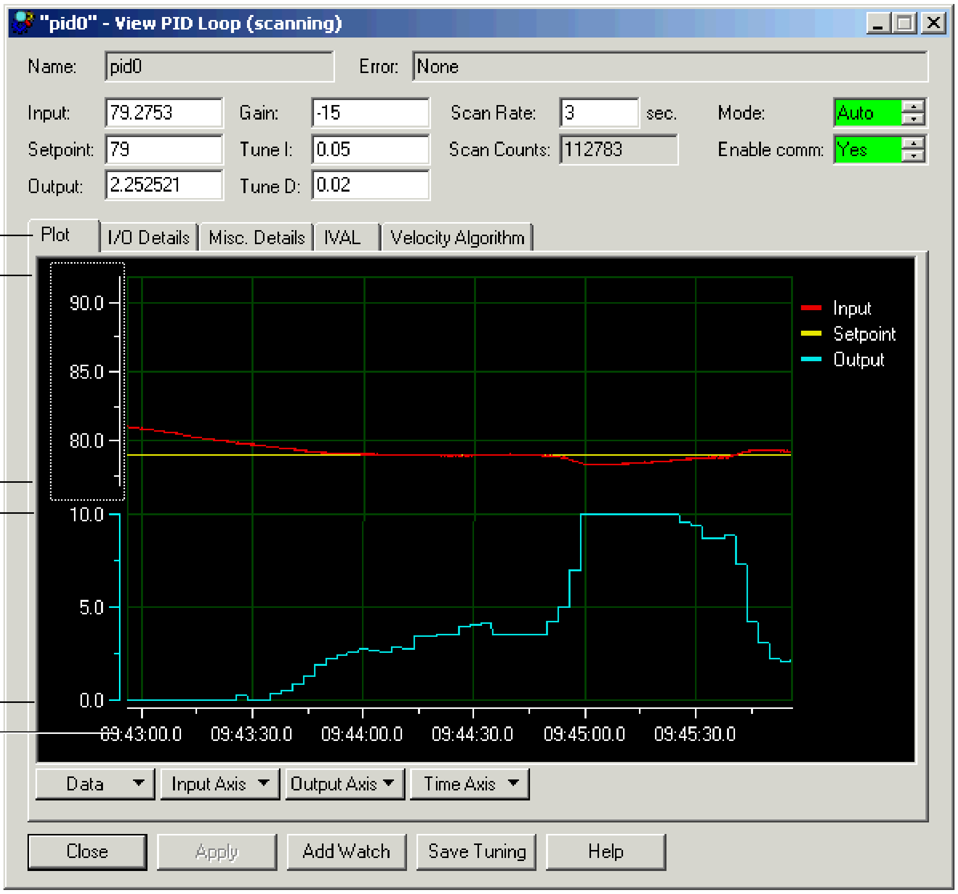
Inspecting and Tuning PID Loops

In Debug mode, you can view PID loops and tune them. This section gives you basic steps for inspecting PIDs, determining system lag, and tuning PIDs. For PIDs on Ethernet-based I/O units, we highly recommend Opto 22 form #1410, *PID Configuration and Tuning: SNAP Ultimate I/O Learning Center Supplement*, which contains more detailed information. Form #1410 is available for download from our Web site at www.opto22.com.

Inspecting a PID (Ethernet)

This section applies to SNAP Ethernet and SNAP Ultimate I/O units only. For serial mistic I/O units, see page 6-71.

1. With the strategy running in Debug mode, double-click the PID on the Strategy Tree.



Name: pid0

Error: None

Input: 79.2753

Gain: -15

Scan Rate: 3 sec.

Mode: Auto

Setpoint: 79

Tune I: 0.05

Scan Counts: 112783

Enable comm: Yes

Output: 2.252521

Tune D: 0.02

Click a tab to see or change additional data.

Setpoint and Input plot. Adjust resolution using the Input Axis button below the plot. Click and drag on the scale to move the line.

Output plot. Adjust resolution using the Output Axis button. Click and drag on the scale to move the line.

Time axis. Adjust resolution using the Time Axis button. Click and drag left or right to see other times.

2. View or change PID parameters as necessary. Click the other tabs to see additional data. To tune the PID, see page 6-64.

3. To add the PID to a watch window, click Add Watch and see page 6-73.

4. To save, copy, or print the current plot, click the Data button and choose from the popup menu.

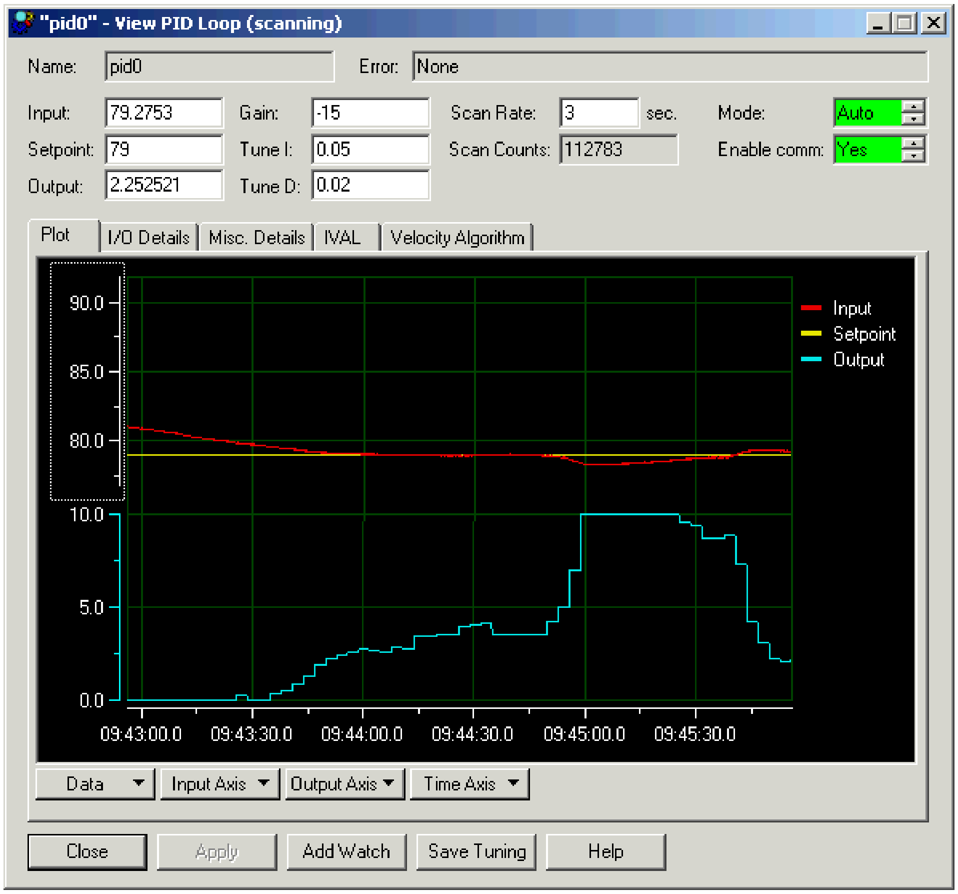
5. To save changes to any of the PID configuration parameters, click Save Tuning.

Determining System Lag

You can directly control the PID output to determine system lag, which is essential to setting the PID scan rate. Also see Opto 22 form #1410, *PID Configuration and Tuning: SNAP Ultimate I/O Learning Center Supplement*, available for download from our Web site at www.opto22.com.

1. Determine two significantly different output settings that are within the capabilities of your system.

2. With the strategy running in Debug mode, double-click the PID on the Strategy Tree.



Name: pid0
Error: None
Input: 79.2753
Gain: -15
Scan Rate: 3 sec.
Mode: Auto
Setpoint: 79
Tune I: 0.05
Scan Counts: 112783
Enable comm: Yes
Output: 2.252521
Tune D: 0.02

3. Set the Mode to Manual (if not set already) and click Apply.

4. In the Output field, type one of your two output settings and click Apply.

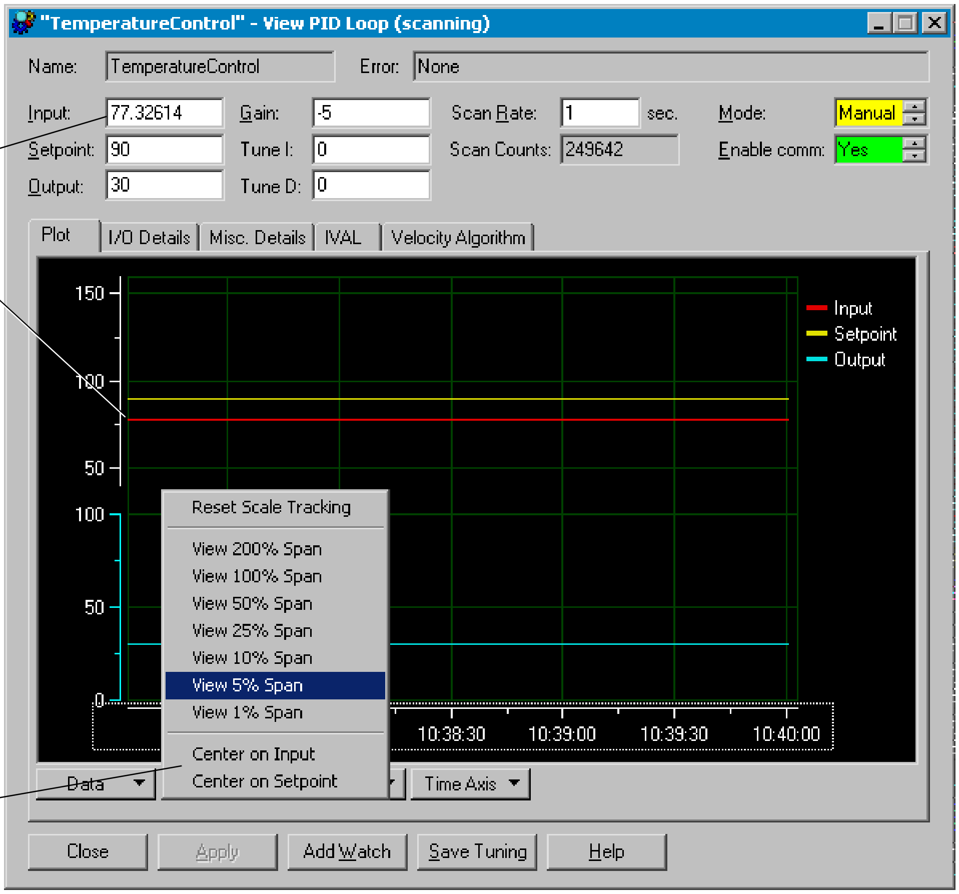
   Use an output value typical of your system but low enough to allow you to change output by 20%.

5. Reset your time axis, if necessary, by clicking the Time Axis and choosing from the popup menu. Then choose Reset Scale Tracking from the same menu.

   The span setting varies according to your system; a 3-minute span is often suitable. Until you are familiar with the PID plot, it is recommended that you avoid using the shortest settings (10-seconds or 1-second). After you've observed a change in the input, you can zoom in on the graph, which is described later.

6. Wait for your system to stabilize.

The system is stable when the Input value does not vary significantly (some drift can be expected). Stabilization may take several minutes, depending on the system.



"TemperatureControl" - View PID Loop (scanning)

Name: TemperatureControl Error: None

Input: 77.32614 Gain: -5 Scan Rate: 1 sec. Mode: Manual

Setpoint: 90 Tune I: 0 Scan Counts: 249642 Enable comm: Yes

Output: 30 Tune D: 0

A stable system exhibits little change in the Input value, which is shown numerically and graphically.

Adjust resolution of the Input Axis by clicking the Input Axis button.

7. Increase the resolution of the Input Axis by clicking the Input Axis menu and choosing a span setting of 1 or 5 percent.

8. Center the Input Axis, if necessary, by clicking the red line at its left end and dragging it up or down until the plot is visible.

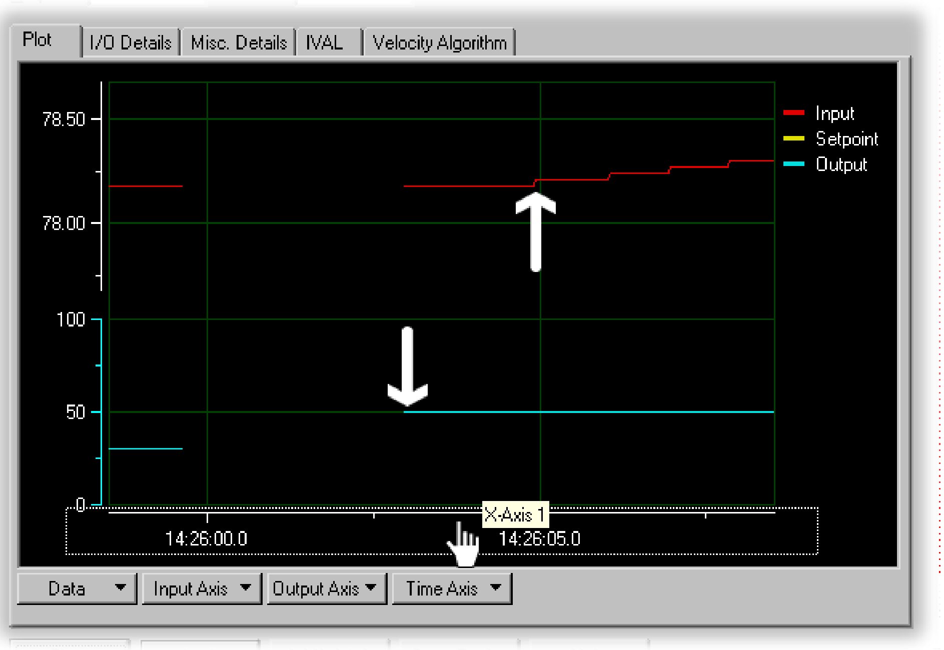
9. Under the Time Axis menu, choose Reset Scale Tracking.

This is a precautionary step, as changing settings on the plot can fix the plot at a certain point in time. Resetting the time axis ensures that you are viewing the real-time values.

10. In the Output field, type the other of your two output settings and click Apply.

A 20% percent increase is a moderate, detectable change for most systems. Your system may require a larger or a smaller change to stay within safety constraints.

11. Wait for a discernible change in the Input axis.

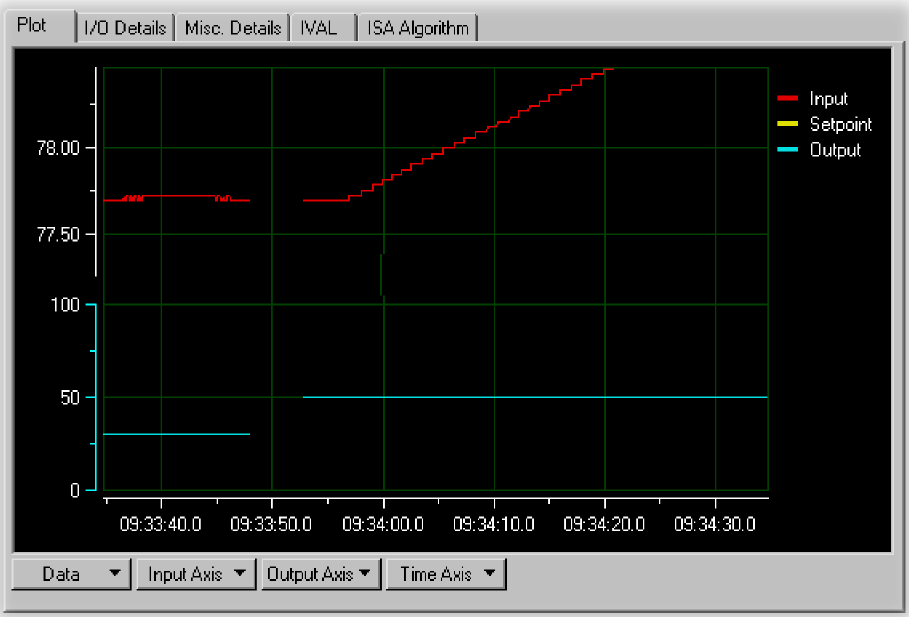


The Input axis (indicated by the upper white arrow added to this picture) begins to respond to the change in output (lower white arrow).

12. Increase the resolution of the Time Axis by clicking the Time Axis button and choosing a lower percentage, such as View 1 Minute Span.

13. Scroll the Time Axis to locate the point at which you changed the Output.

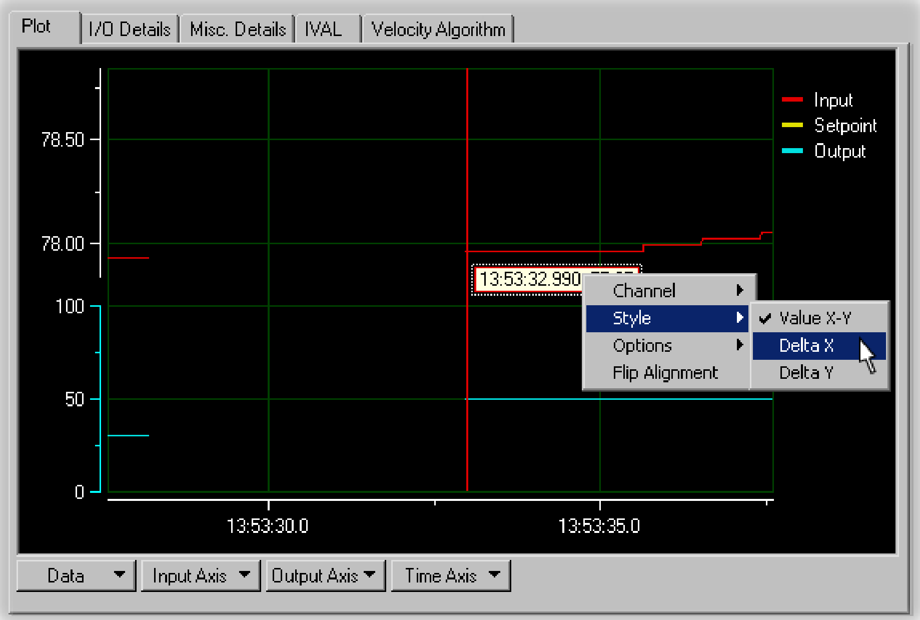
Both the time and input axes should display the point at which the Output changed, the lag, and the point the input changed. If not, adjust the Input axis and Time axis, until this information is displayed.



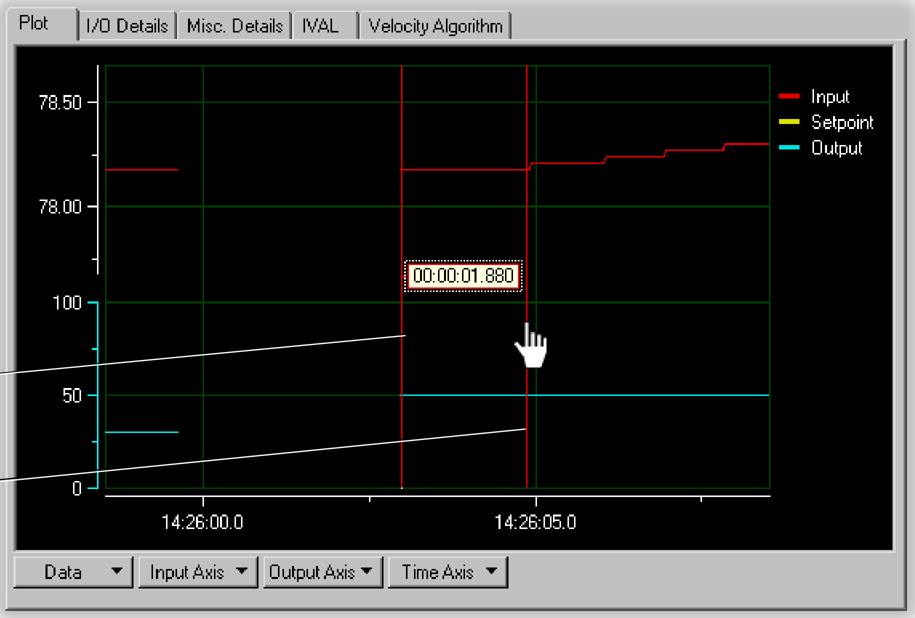
14. From the Data menu, select Cursor.

The data cursor, a line with a value bar attached to it, appears on the plot.

15. Right-click the data cursor and choose Delta X from the Style submenu, as shown below.



16. To measure the system lag (the time between the change in output and the change in input), click and drag the first vertical red bar to just after the output change. Drag the second bar to just before the change in input.



The Delta X cursor displays the time difference between the two vertical bars.

Drag the first bar into position after the Output change.

Drag the second bar to a position just before the change in Input.

If you are unable to get the precision you want, you can view the plot at a lower time span, such as 10 seconds. You will need to reposition your plot and the measurement bars of the Delta X cursor.

The example above shows a system lag of 1.88 seconds. Generally, a suitable scan rate can be anywhere from one-third the system lag to two times the system lag. Considerations in setting scan rate are:

- Slower scan intervals may be easier to tune. The PID controller has time to see the effect of the previous output before calculating a new output.
- Faster scan rates may be necessary to achieve the desired response. When scan intervals are shorter than the system lag, tuning must compensate for any over-correction from the controller output.

Tuning a PID Loop (Ethernet)

NOTE: This section applies to SNAP Ethernet and SNAP Ultimate I/O units only. For information on tuning PID loops on serial-based mistic I/O units, see "PID—Mistic Commands" on page 10-62.

Tuning a PID involves manipulating the P, I, and D constants in real time. The following steps should be viewed as general suggestions to show you features that are available for tuning. We highly recommend Opto 22 form #1410, *PID Configuration and Tuning: SNAP Ultimate I/O*

Learning Center Supplement, for more detailed information. Form #1410 is available for download from our Web site at www.opto22.com.

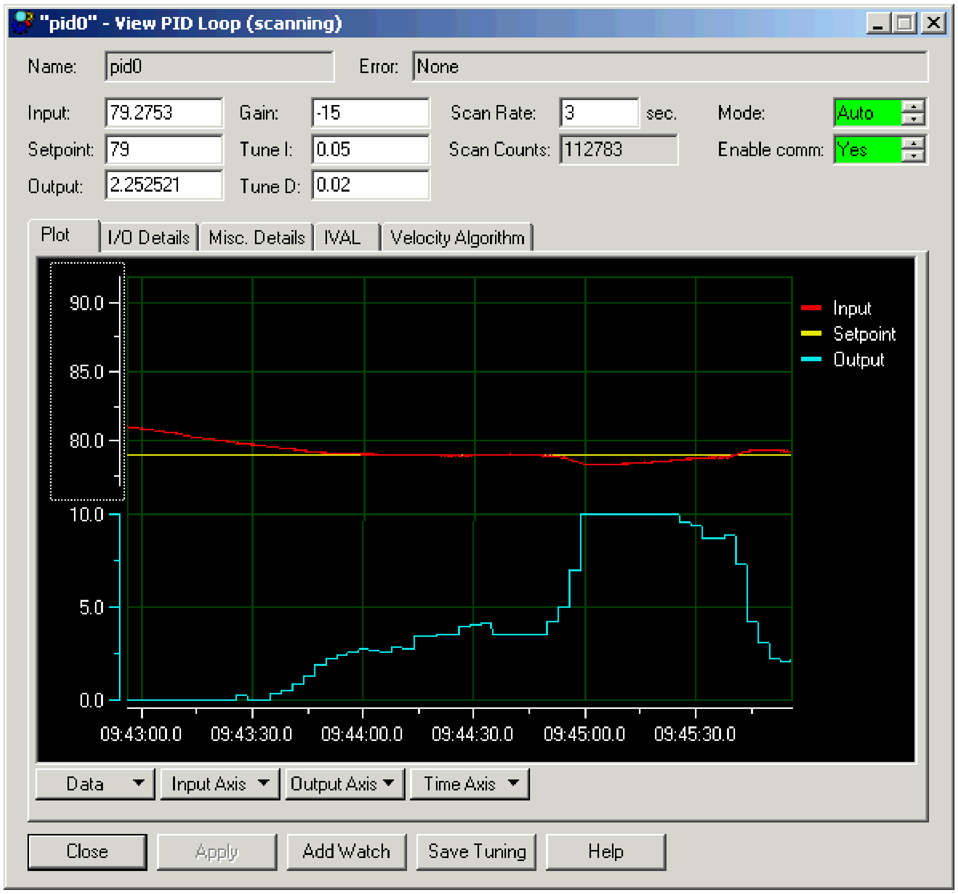
CAUTION: Before following these procedures, make sure you know the limits of the equipment being controlled and monitored by your PID loop. Also, make sure that these points are configured properly. Any values suggested in these steps are for example only and must be modified according to the capabilities and constraints of your system.

1. Make sure the following PID features have already been configured:
   - Scan Rate
   - Input
   - Input low range and high range
   - Output
   - Output lower and upper clamp.
   - Algorithm
   - Setpoint. If your setpoint changes during normal operation, tune your PID with the setpoint configured to host, so you can simulate setpoints from an Input Point or from another PID.
   - Gain. A final gain constant will be determined by tuning, but before you can tune your PID, your gain constant must be either a positive or negative number according to the type of system you have. For example, a heating system reports a negative error when heat needs to be applied; a negative gain constant turns this error into a positive output for the heater. Alternatively, a cooling system reports a positive error when the input exceeds the setpoint; a positive gain constant maintains the positive output to the chiller.
   - Optional, depending on your system: Minimum and maximum changes to Output and Output forcing when the Input is out of range.

2. Download and run your strategy.

   The current PID configuration is written to the I/O unit. You can stop your strategy at this point if you wish, as the PID will continue operating.

3. In Debug mode, double-click the PID on the Strategy Tree.



Name: pid0
Error: None
Input: 79.2753
Gain: -15
Scan Rate: 3 sec.
Mode: Auto
Setpoint: 79
Tune I: 0.05
Scan Counts: 112783
Enable comm: Yes
Output: 2.252521
Tune D: 0.02

4. Set the PID Mode to Auto (if not set already) and click Apply.

5. Change the Setpoint, if desired, by typing a new setpoint and clicking Apply. (Setpoint must be configured as Host.)

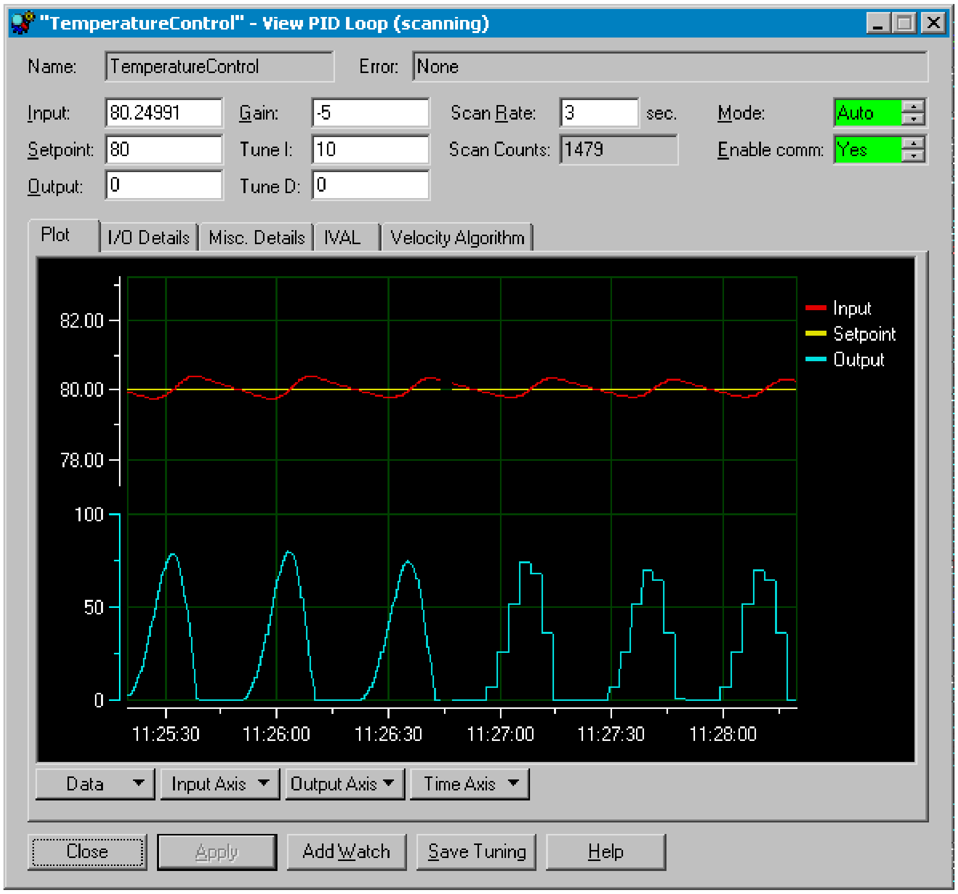
   Depending on the type of system, your PID may maintain a setpoint or respond to changes in setpoint. Experiment with setpoint changes again after tuning the P, I, and D constants.

6. Adjust the span of the input, output, and time axes according to how much change you expect from your system. To set a span, click the axis button and choose from the popup menu.

7. If desired, type a new Scan Rate and click Apply.

   For most systems, you should use an appropriate scan rate based on the system lag (see "Determining System Lag" on page 6-59). However, you can experiment with Scan Rates before tuning the P, I, and D constants or adjust scan rate after tuning.

Here is an example for Scan Rate:



"TemperatureControl" - View PID Loop (scanning)

Name: TemperatureControl

Error: None

Input: 80.24991

Gain: -5

Scan Rate: 3 sec.

Mode: Auto

Setpoint: 80

Tune I: 10

Scan Counts: 1479

Enable comm: Yes

Output: 0

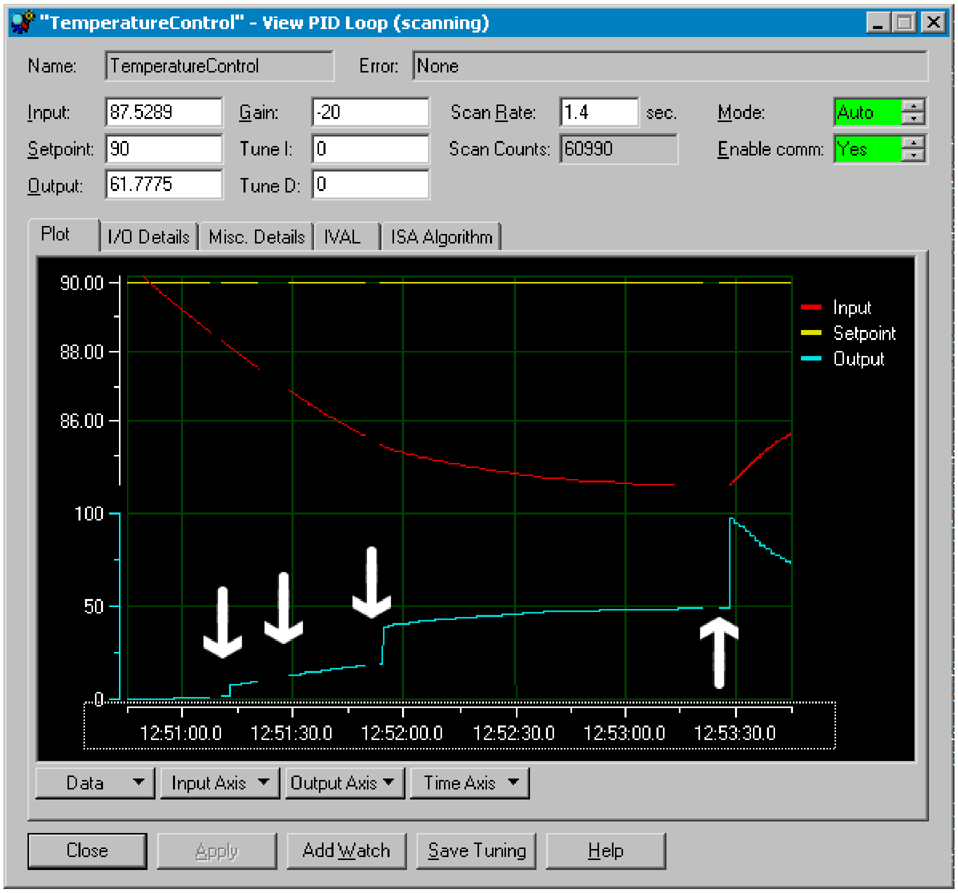
Tune D: 0

The lag for this system was determined to be about 2 seconds. The left half of the plot reflects a 0.5-second scan rate, while the right half shows a 3-second scan rate. Notice both scan rates have the same effect on the input; however, the 3-second scan rate is using less of the processor's resources.

8. Experiment with gain settings by typing a new value in the Gain field and clicking Apply.

The easiest way to tune most PIDs is to experiment with the gain constant first. Try various gains to see how well the system stays at setpoint and responds to setpoint changes.

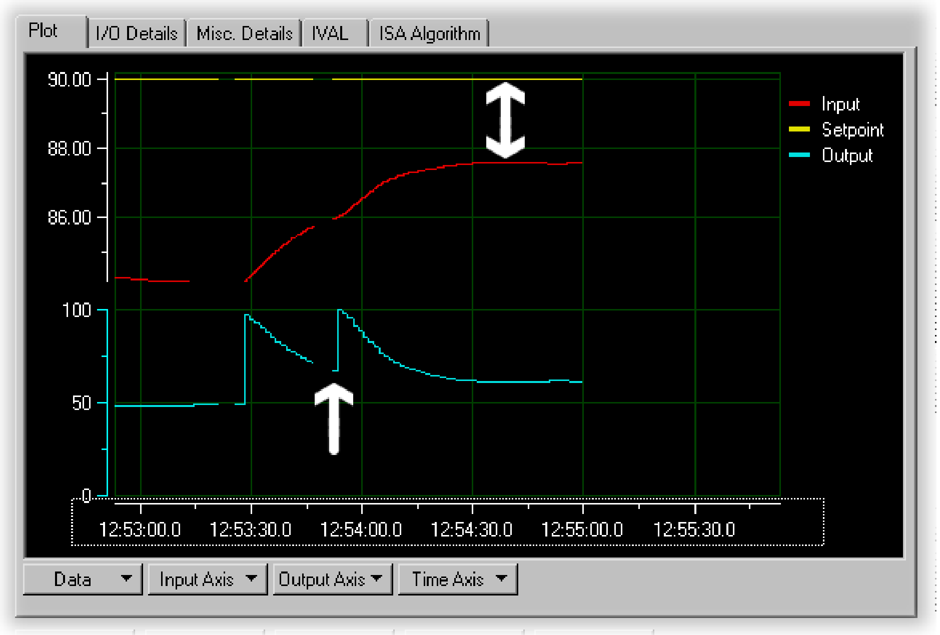
In the example below, the white arrows (added for the example), show where gain constants of -2, -5, -10, and --20 were applied:



Name: TemperatureControl
Error: None
Input: 87.5289
Gain: -20
Scan Rate: 1.4 sec.
Mode: Auto
Setpoint: 90
Tune I: 0
Scan Counts: 60990
Enable comm: Yes
Output: 61.7775
Tune D: 0

In this example, a gain setting of -30 revealed an offset error:

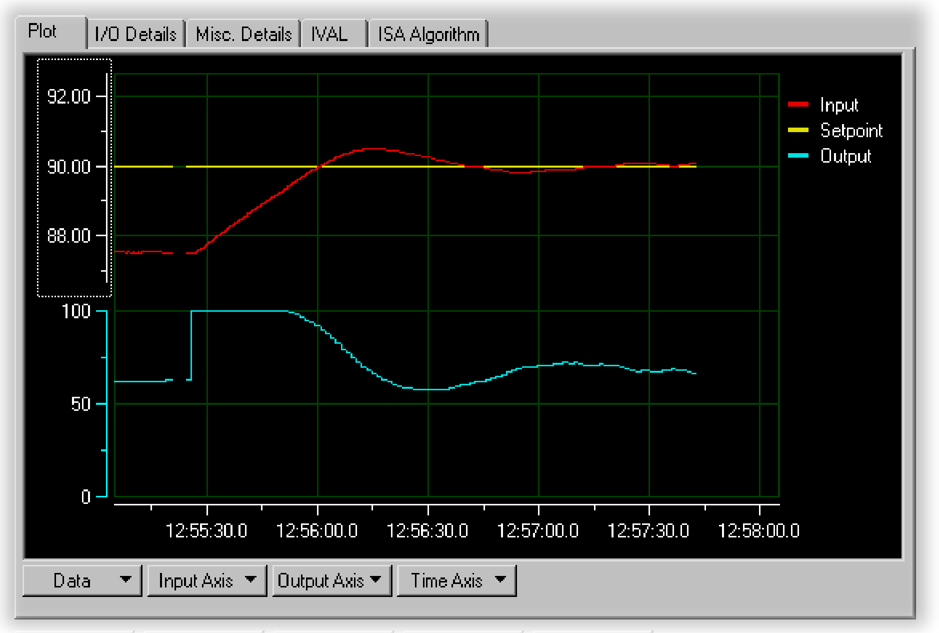
With only a gain constant applied, the input often stabilizes at an incorrect value. In this heating example, a gain setting of -30 drove the input close to the setpoint, but subsequent increases failed to eliminate the offset. It is time to try integral constants to eliminate the offset error.



9. Experiment with the integral constant: in the Tune I field, type an number between 0 and 1 and click Apply. (Your PID may require larger numbers.)

   In this example, an integral constant of 0.1 corrected the offset error.

The far left side of the plot shows the offset before an integral constant of 0.1 was applied. This setting eliminated the offset. In many applications, a minor fluctuation around the setpoint is acceptable, and these applications use gain and integral only.In some applications, however, the fluctuations at the setpoint indicate that the gain is too high (too much gain makes a system unstable) or that a derivative constant is required.

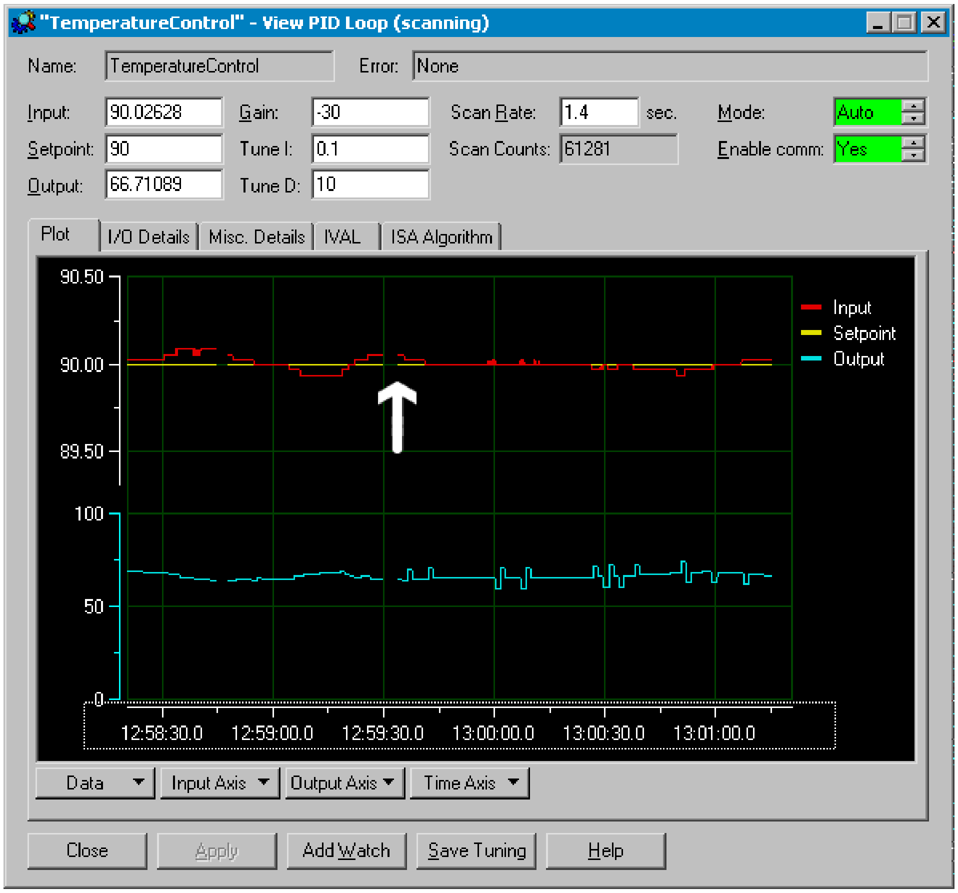


10. If derivative correction is needed, experiment with the derivative constant: in the Tune D field, type 1 and click Apply. (Your PID loop may require a larger number.)

In this example, a derivative of 10 makes a noticeable difference in keeping the input near the setpoint.

Many PID systems are effectively controlled with gain and integral constants only and no derivative constant. In this example, the gain and integral settings are maintaining the temperature at 0.06 from setpoint. To demonstrate the effect of the derivative constant, the resolution of the input axis was increased to show a 1 percent span. At this resolution, the plot reveals changes of 0.01 degrees F.

The left side of the plot shows the effect of gain at -30, integral at 0.1, and no derivative constant. The arrow shows when a derivative constant of 10 was applied. The right side of the plot shows how the derivative constant is keeping the input closer to setpoint.



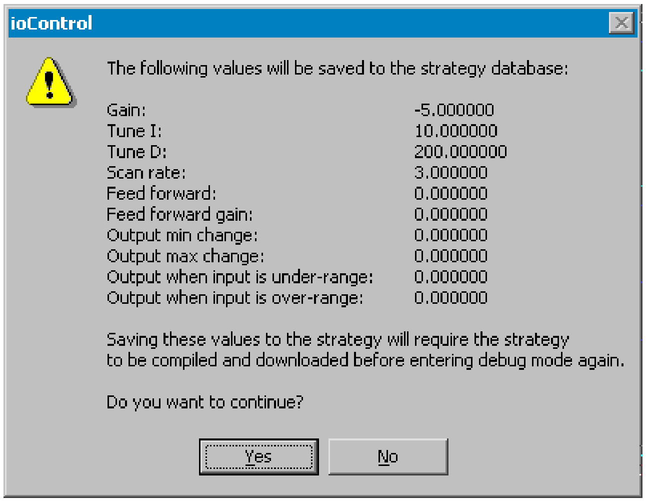
"TemperatureControl" - View PID Loop (scanning)

Name: TemperatureControl
Error: None

Input: 90.02628
Setpoint: 90
Output: 66.71089
Gain: -30
Tune I: 0.1
Tune D: 10
Scan Rate: 1.4 sec.
Scan Counts: 61281
Mode: Auto
Enable comm: Yes

11. Click Save Tuning to save your tuning parameters to the strategy database.

Changes are lost unless you save them. You may wish to save your tuning parameters when you see any improvement in performance, even if they are not final.



The following values will be saved to the strategy database:

Gain:	-5.000000
Tune I:	10.000000
Tune D:	200.000000
Scan rate:	3.000000
Feed forward:	0.000000
Feed forward gain:	0.000000
Output min change:	0.000000
Output max change:	0.000000
Output when input is under-range:	0.000000
Output when input is over-range:	0.000000

Saving these values to the strategy will require the strategy to be compiled and downloaded before entering debug mode again.

Do you want to continue?

12. Click Yes.

Values are saved to the ioControl strategy. Remember to save PID parameters to the I/O unit's flash memory, too (see "Inspecting I/O Units" on page 6-52).

Inspecting a PID Loop (mistic)

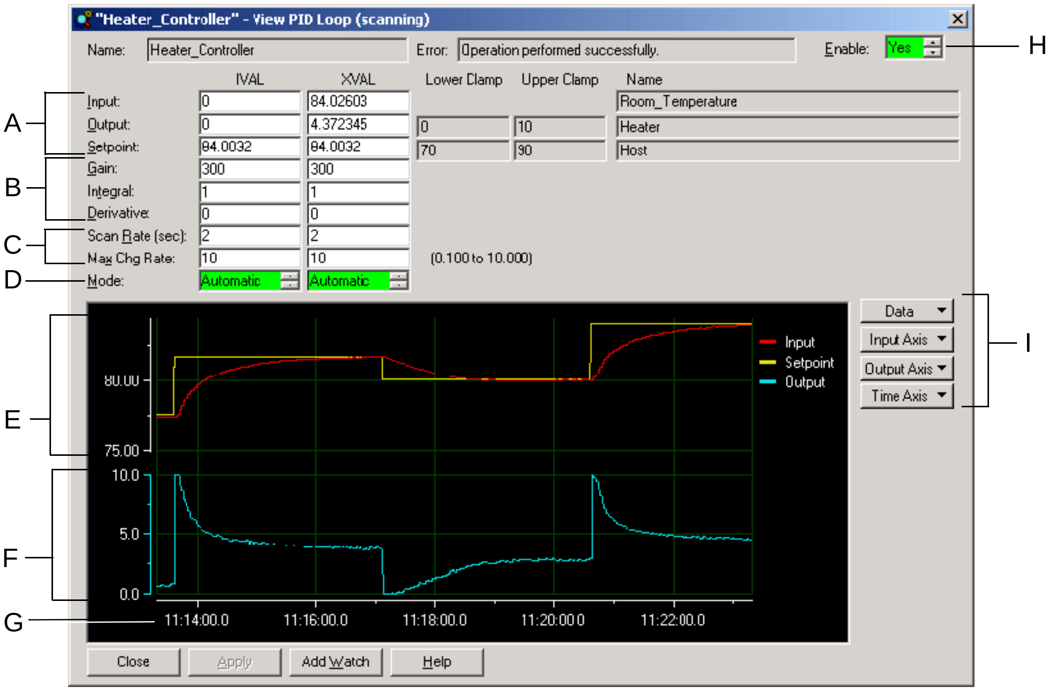
Pro

(ioControl) This section applies to serial mistic I/O units only. For SNAP Ethernet and SNAP Ultimate I/O units, see page 6-58.

You can review a PID loop's data, modify its status, or set its Internal Values or External Values in Debug mode. To monitor the PID loop in a watch window, see page 6-73. To change the PID loop, follow these steps.

1. With the strategy running in Debug mode, double-click the PID loop on the Strategy Tree, or double-click it in the View I/O Unit dialog box.

The View PID Loop dialog box appears, showing the configuration parameters for the PID loop and several values you can change. The title bar shows the name of the PID loop and whether scanning is occurring.



"Heater_Controller" - View PID Loop (scanning)

Name: Heater_Controller
Error: Operation performed successfully.
Enable: Yes

	IVAL	XVAL	Lower Clamp	Upper Clamp	Name
Input:	0	84.02603			Room_Temperature
Output:	0	4.372345	0	10	Heater
Setpoint:	84.0032	84.0032	70	90	Host
Gain:	300	300			
Integral:	1	1			
Derivative:	0	0			
Scan Rate (sec):	2	2			
Max Chg Rate:	10	10	(0.100 to 10.000)		
Mode:	Automatic	Automatic			

Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

Asterisks in a field indicate an out-of-range value. Dashes in an XVAL field indicate a communication error.

2. Change the fields as necessary. See "View PID Loop (mistic) Dialog" below.

3. To add the PID loop to a watch window, click Add Watch and see page 6-73.

4. To save, copy, or print the current plot, click the Data button and choose from the popup menu.

View PID Loop (mistic) Dialog

(A) Input, Output, and Setpoint (current internal and external values) Change each to any valid value. If lower and upper clamps appear to the right of the value, these clamps define the range of valid values. Otherwise, the valid range is defined by the I/O point itself. If you can change them, click Apply.

(B) Gain, Integral, and Derivative (current internal and external values) You can change the gain term to any value except zero in the range -32768 to 32767. You can change the integral and derivative terms to any value in the range zero to 32767. If you change them, click Apply.

(C) Scan Rate and Maximum Change Rates (current internal and external values) You can change the scan rate to any value in the range 0.1 to 6553.5 seconds. You can change the maximum change rate to a value between one percent and 100 percent of the output range (defined by the output's zero-scale and full-scale values). If you change them, click Apply.

Asterisks in an IVAL field indicate that a valid scan rate or maximum change rate hasn't been read yet (usually before the strategy is run).

(D) PID execution mode A green background means Automatic, a yellow background means Manual. Click an arrow to change the mode; then click Apply.

(E) Setpoint and Input plot Adjust resolution using the Input Axis button at right. Click and drag on the scale to move the line.

(F) Output plot Adjust resolution using the Output Axis button. Click and drag on the scale to move the line.

(G) Time axis Adjust resolution using the Time Axis button. Click and drag left or right to see other times.

(H) Current status Yes on a green background means enabled; No on a red background means disabled. To change status, click one of the arrows; then click Apply.

(I) Data and Axis buttons

Using Watch Windows for Monitoring

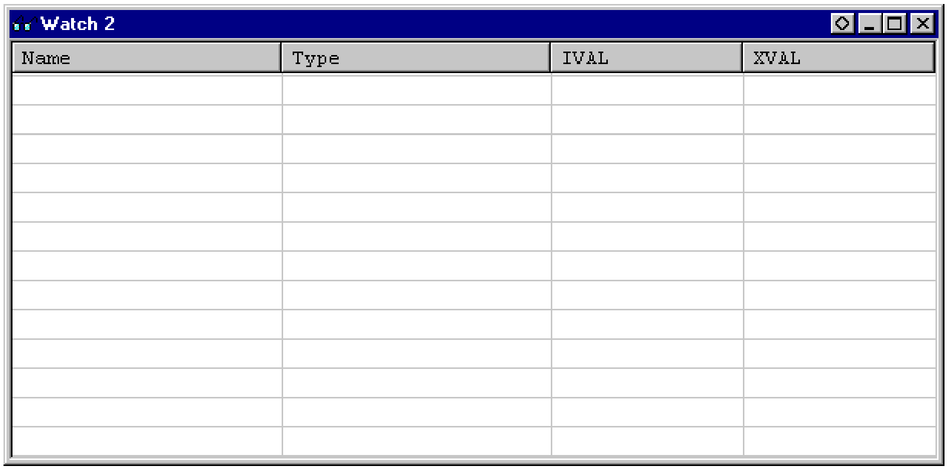
While the Strategy is running, you can monitor several strategy elements at once in a watch window: I/O units, digital and Analog Points, PID loops, variables, even charts. You cannot monitor subroutine parameters or variables that are local to a subroutine in a watch window.

Unlike inspection windows, watch windows can be created the way you want, docked in a position most convenient for you, and are saved with your strategy. You cannot change strategy elements in a watch window, but you can open the inspect dialog box from the watch window and change the element there.

Creating a Watch Window

1. With the strategy open and in Debug mode, click the New Watch Window icon in the toolbar or choose Watch→New.
2. In the Create New Watch Window dialog box, navigate to the location where you want the watch window file to be kept (usually in the same folder as the strategy). Enter the watch window file name and click Open.

The empty watch window appears.



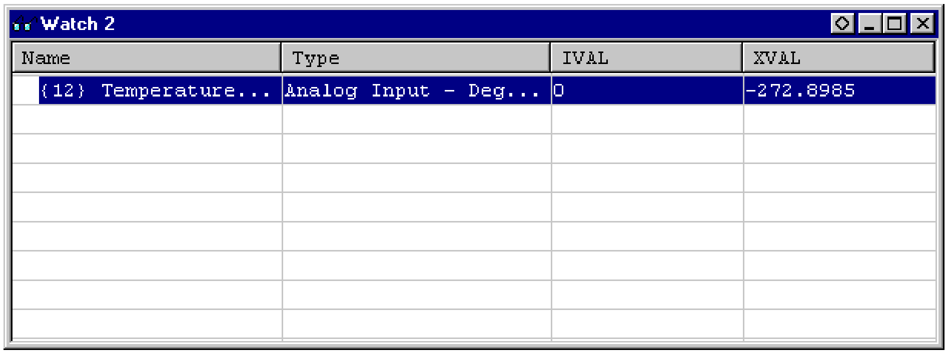
Watch 2

Name	Type	IVAL	XVAL

3. Add elements you want to watch in this window by clicking them on the Strategy Tree and dragging them into place in the watch window, or by right-clicking them and choosing Watch from the pop-up menu.

   You can add I/O units, digital and Analog Points, PID loops, variables, and charts. You cannot add subroutine parameters or variables that are local to a subroutine.

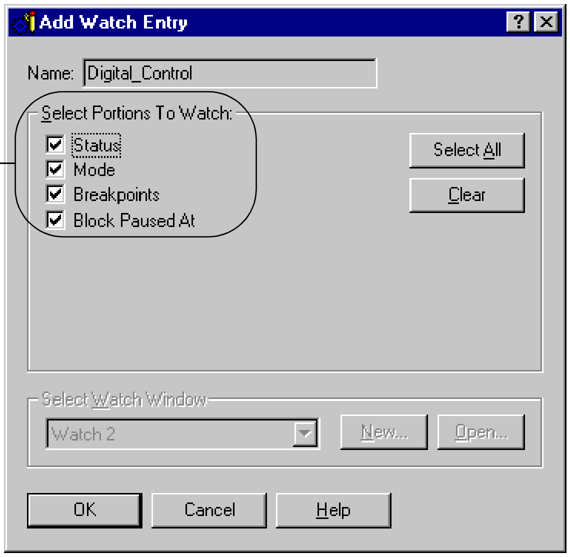
   Depending on which element you add and how you add it, it may appear immediately in the window, as shown here.



Watch 2

Name	Type	IVAL	XVAL
{12} Temperature...	Analog Input - Deg...	0	-272.8985

For some elements, the Add Watch Entry dialog box appears, so you can specify what to watch.



Add Watch Entry

Name: Digital_Control

Select Portions To Watch:

☑ Status
☑ Mode
☑ Breakpoints
☑ Block Paused At

Select Watch Window: Watch 2

Items in this area vary depending on the element you are watching. This example shows a chart.

4. If an Add Watch Entry dialog box appears, click to place or remove the check mark next to any item. When all the items you want to watch are checked, click OK.

   The element is added to the watch window.

   The watch window is automatically saved.

Opening an Existing Watch Window

If a watch window was open when you exited Debug mode, it will automatically open again when you re-enter Debug mode. To open other watch windows, follow these steps.

1. Make sure the strategy is open and in Debug mode.
2. Click the Open Watch Window icon on the toolbar, or choose Watch→Open.
3. Navigate to the watch window you want to open and double-click its name.

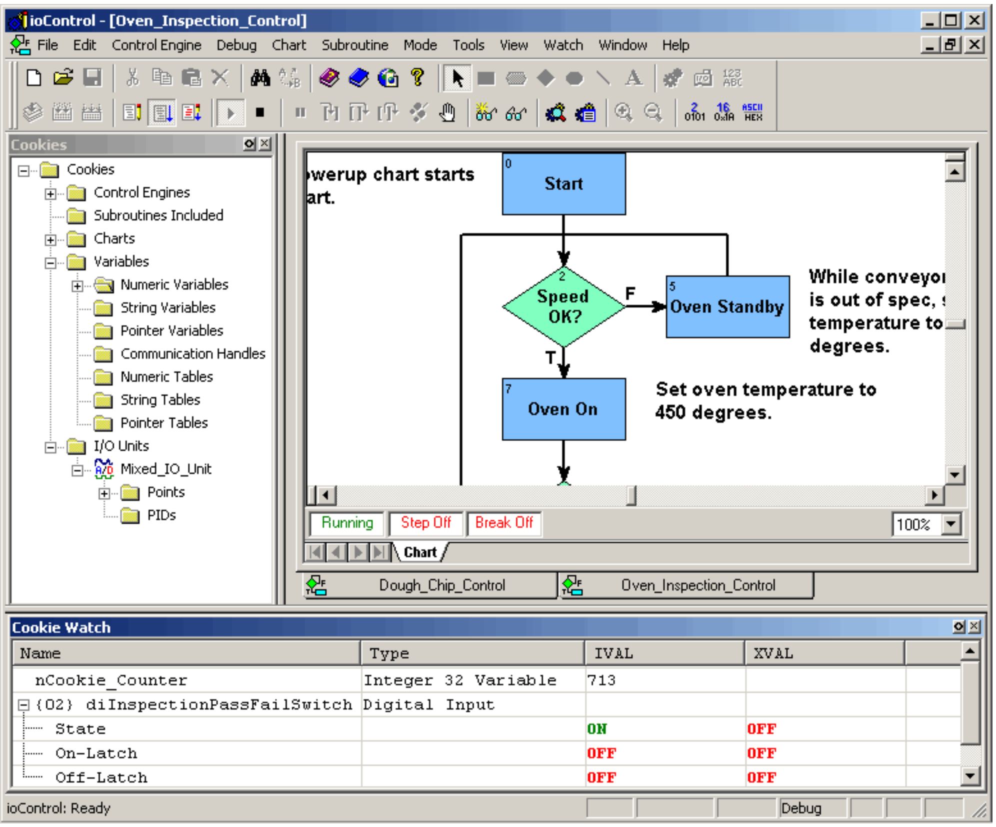
   The window opens in the position you left it.

Working in Watch Windows

Watch windows are flexible. You can dock the window where you want it in the ioControl main window. You can also move, delete, and inspect elements in the window.

- **To dock the watch window**, click the docking icon in its title bar.

  The window moves to its own frame.



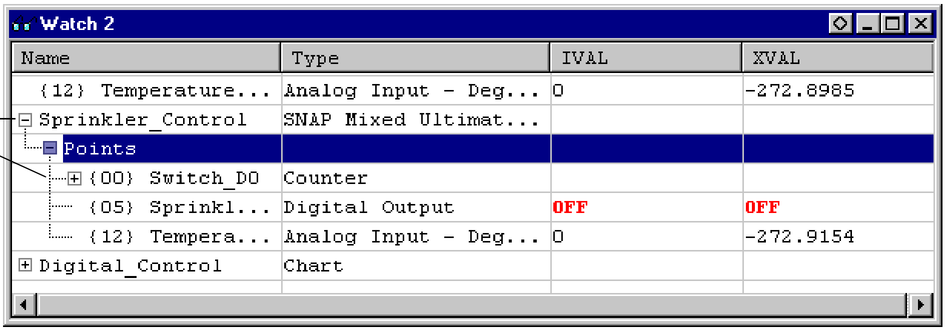
Cookie Watch

Name	Type	IVAL	XVAL
nCookie_Counter	Integer 32 Variable	713	
⊟ {02} diInspectionPassFailSwitch	Digital Input		
State		ON	OFF
On-Latch		OFF	OFF
Off-Latch		OFF	OFF

"Docked" watch window

See "Docking Windows" on page 3-17 for more information.

- **To expand or collapse watch window** you have added an I/O unit, PID loop, chart, or table to, click its plus or minus sign.



Watch 2

Name	Type	IVAL	XVAL
{12} Temperature...	Analog Input - Deg...	0	-272.8985
⊟ Sprinkler_Control	SNAP Mixed Ultimat...		
⊟ Points			
⊞ {00} Switch_DO	Counter		
{05} Sprinkl...	Digital Output	OFF	OFF
{12} Tempera...	Analog Input - Deg...	0	-272.9154
⊞ Digital_Control	Chart		

Expand or collapse the item by clicking the + or - sign in the box.

- **To rearrange elements** in the watch window list, click the item you want to move and drag it to a new location, or right-click it and choose Move Up or Move Down from the pop-up menu.

  You can also sort elements in the window by clicking on the column label. For example, to sort by Type, click the label Type in the column heading. Click again to change the order from ascending (A–Z) to descending (Z–A).

- **To move an element** from one watch window to another, open both windows and drag the element where you want it. To copy an element to another watch window (so it will appear in both windows), hold down the CTRL key while you drag it.
- **To delete an element**, right-click it and choose Delete from the pop-up menu.
- **To inspect an element**, double-click it.

  The inspect dialog box opens. For information on using it, see "Inspecting Control Engines and the Queue" on page 5-12.

CHAPTER 7

Working with Strategies

Introduction

A strategy is the software program you create in ioControl. A strategy is similar to a file in any Microsoft Windows program. You use standard Windows menu items to create a new strategy, to open an existing strategy, or to save a strategy. The strategy includes all the definitions and instructions necessary to control your process. This chapter is a step-by-step reference for working with strategies in all three strategy modes: Configure, Debug, and Online.

In this Chapter

Creating a New Strategy

Each ioControl Strategy must be located in its own directory. When you create a new strategy you must create a new directory or use an empty one. Having each strategy in its own directory keeps all its files in one place and makes it easy to copy a strategy to another location for modification or backup.

1. To create a new strategy, select File➞New Strategy, or press CTRL + N, or click the New Folder button on the toolbar.

2. In the New Strategy dialog box, navigate to the directory where you want the strategy to be placed. Create a new folder if necessary.

3. Type the strategy name.

   As you can see in the Files of type field, ioControl files have an extension of .idb.

4. Click Open.

   The new strategy is created. Its Strategy Tree and Powerup charts appear in the ioControl main window. For information on the main window, see page 3-11. For programming information, see Chapter 4, “Designing Your Strategy.” For steps to create charts, Chapter 8, “Working with Flowcharts.”

Opening a Strategy

Only one strategy at a time can be open in ioControl. If you currently have a strategy open, it must be closed before another is opened. You are prompted to save changes before it closes.

Opening an Existing Strategy

1. To open an existing strategy, select File→Open Strategy, or press CTRL + O, or click the Open Strategy button 📂 on the toolbar.

2. In the Open Strategy dialog box, navigate to the strategy you want to open and click Open.

   The strategy opens in Configure mode, with the windows in the same position they were when the strategy was closed.

Opening a Recently Used Strategy

To open a strategy you have recently used, choose its name from the list at the bottom of the File menu. The ten most recently opened strategies are listed.

Loading a Strategy or Mode at Startup

To have ioControl automatically start up with the strategy that was open when you exited, choose Configure→Options and click to put a check mark next to Load Last Strategy at Startup.

To have ioControl open strategies in the same mode as when you exited ioControl, choose Configure→Options and click to put a check mark next to Load Last Mode at Startup.

Opening Strategies in ioControl Basic and ioControl Professional

A strategy saved in ioControl version 6.1 or less can be opened in either ioControl Basic or Professional. An ioControl Basic strategy can also be opened in ioControl Professional. However, ioControl Professional strategies cannot be opened in Basic or in any earlier version of ioControl.

CAUTION: *Once a strategy is opened in ioControl Professional, it can no longer be opened in ioControl Basic.*

Opening an OptoControl Strategy

If you are moving a Strategy from OptoControl to ioControl Professional, ioControl will open it and help you convert it. Although many things will convert without difficulty, planning ahead is essential to make the job easier. **Before opening an OptoControl strategy in ioControl,** read the *FactoryFloor to ioProject Migration Technical Note*, Opto 22 form #1596.

Saving and Closing

CAUTION: *Once a strategy is opened in ioControl Professional, it can no longer be opened in ioControl Basic.*

Saving the Strategy and All Charts

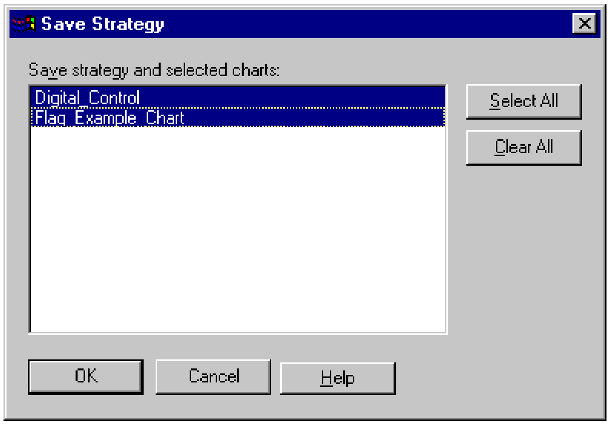
To save all your work quickly, choose File→Save All. The strategy and all modified charts and subroutines are saved.

Saving the Strategy and Some Charts

NOTE: You cannot save changes to a subroutine this way. To save a subroutine, use File→Save All, or use Subroutine→Save or Subroutine→Save All.

1. To save changes to some charts but not others, click the Save Strategy button 💾 on the toolbar (or choose File→Save Strategy, or press CTRL + S).

   The Save Strategy dialog box appears, highlighting all charts modified since the last save. In this example, two charts have been modified:



Digital_Control
Flag_Example_Chart

2. To save some charts and not others, press CTRL and click any charts you don't want to save.

   You can also click Clear All to select none of the charts, or click Select All to select all of the charts.

3. When only the charts you want to save are highlighted, click OK.

   The strategy and the highlighted charts are saved.

Saving the Strategy to a New Name

1. To save the strategy and all its charts under a new name, choose File→Save Strategy As.
2. In the Save Strategy As dialog box, navigate to where you want the new strategy to be. Create a new folder if necessary.

   Remember that each strategy must be in its own directory.

3. In the Strategy Name field, enter the new strategy name. Click Save.

   The strategy and all its charts are saved under the new name in the new directory.

Saving Before Debugging

When you change to Debug mode, you are prompted to save a Strategy you have modified. If you don't want to be prompted to save before entering Debug mode, choose Configure→Options and click to remove the check box for Prompt To Save Strategy Before Running Debugger.

Closing a Strategy

To close a Strategy, click the close box in the Strategy Tree or choose File→Close Strategy.

NOTE: Since only one strategy at a time can be open in ioControl, creating a new strategy or opening an existing strategy automatically closes any current strategy first. If you've made changes to the current strategy, you are prompted to save them.

Saving a Strategy to Flash

When you finish working on your Strategy and have downloaded it, you should save it to the control engine's flash memory. By default, a strategy is downloaded to the control engine's RAM. Saving it to flash protects the strategy in case of a power loss. You can save it to flash just once, when needed, or save every time the strategy is downloaded.

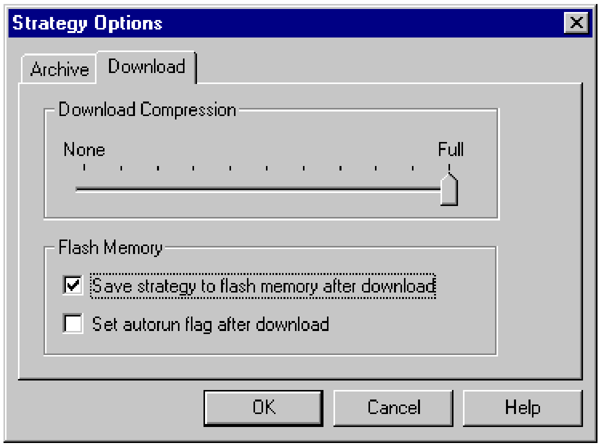
Saving to Flash Once

You can save the strategy to flash at any time when you are in Debug mode. To do so, choose Control Engine→Save Strategy to Flash.

Saving to Flash on Every Download

CAUTION: *It is possible to wear out flash memory if you save to it many, many times. Use the following steps only when your strategy is finished.*

1. When you have finished the strategy and are in Configure mode, choose File→Strategy Options.
2. In the Strategy Options dialog box, click the Download tab. Check Save strategy to flash memory after download.



☑ Save strategy to flash memory after download

□ Set autorun flag after download

If a control engine loses power and then restarts, the autorun flag tells the control engine to automatically start running the strategy that is in flash memory. If the autorun flag is not set, the strategy must be started manually after power is restored to the control engine.

3. To have the strategy run automatically after a control engine restarts, check Set autorun flag after download.
4. Click OK.

Archiving Strategies

Strategy archives help you track changes during development and provide a backup in case of a failure on the control engine or on the computer where the original files are kept. Archive files are date and time stamped, and zipped for compact storage. We recommend you archive both to the computer (during strategy development) and to the control engine (when the strategy is completed).

Archiving to the Computer

Archiving strategies to the computer is an excellent way to track changes over time and to produce a zipped file you can copy to another computer or disk for backup. Archives are always placed in the same folder as your strategy. Since a new archive file is created each time you archive a strategy, remember to manually delete any old archive files you do not want to keep.

ioControl offers three ways to archive strategies to the computer:

- To make an archive at any time, choose File→Archive Strategy. A dialog box shows you the name and location of the archive file.
- To have an archive automatically created whenever the strategy is closed, choose File→Strategy Options. In the Strategy Options dialog box, click Archive strategy to disk when strategy is closed.
- To have an archive automatically created whenever the strategy is downloaded, choose File→Strategy Options. In the Strategy Options dialog box, click Archive strategy to disk when strategy is downloaded.

The archive file name will be in one of the following formats:

Archive method	File name format
Manual archive or archive when strategy is closed	Path\Filename.Archive.D02282002.T114351.zip
Archive on download	Path\Filename.Download.D02282002.T114351.zip
Archive when downloading from online mode	Path\Filename.Online.D02282002.T114351.zip

The date stamp (D) is in the format mm/dd/yyyy. In the examples above, the date is February 28, 2002. The time stamp (T) is in the format hh/mm/ss. In the examples above, the time is 51 seconds past 11:43 A.M.

Archiving to the Control Engine

When you archive a Strategy to the control engine, you are placing the zipped file in battery-backed RAM. If power to the control engine is lost, the archive is still there. Archiving to the control engine as well as the computer makes sure that an older strategy can always be found and updated, even after personnel changes occur and years pass.

Make sure there is sufficient memory in the control engine for the archive file. Battery-backed RAM holds 256KB total; in addition to the archived strategy, it stores persistent variables and variables that are initialized on strategy download.

Steps for archiving to the control engine are on page 5-17.

Compiling and Downloading

Before your Strategy can be tested or run, it must be compiled and then downloaded to a control engine. When a strategy is compiled, all the commands, OptoScript code, charts, and variable and I/O definitions it contains are verified and converted into a format that the control engine can understand. Then the strategy can be sent (downloaded) to a control engine. Only compiled strategies can be downloaded.

NOTE: Before you can download your strategy, make sure you have downloaded the latest firmware to your control engine. For instructions, see the controller's user's guide.

Compiling and Downloading in One Step

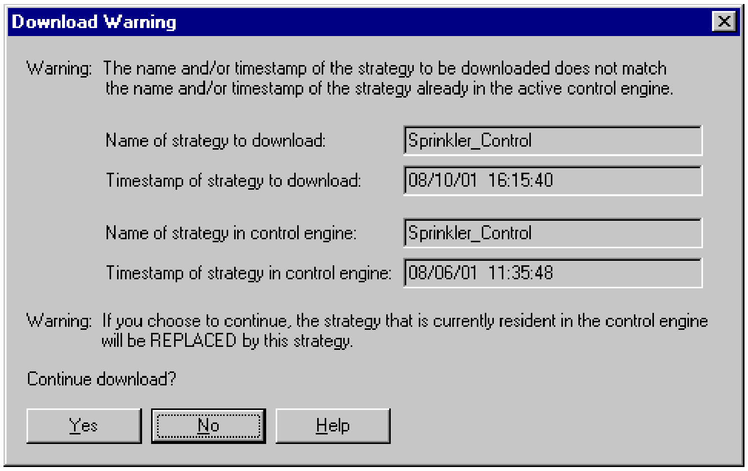
NOTE: If you are using Ethernet link redundancy in ioControl Professional, make sure you are downloading through the IP address you want to use, either the primary or secondary address. See "Using Ethernet Link Redundancy in ioControl" on page 5-6 for more information.

1. With the strategy open in ioControl, click the Debug Mode button on the toolbar, or choose Mode→Debug.

   Changing to Debug mode automatically saves and compiles the strategy, including all code in OptoScript Blocks.

2. If you see a Powerup Clear Expected message, click OK.

   A download warning message may appear.



Download Warning

Warning: The name and/or timestamp of the strategy to be downloaded does not match the name and/or timestamp of the strategy already in the active control engine.

Name of strategy to download: Sprinkler_Control

Timestamp of strategy to download: 08/10/01 16:15:40

Name of strategy in control engine: Sprinkler_Control

Timestamp of strategy in control engine: 08/06/01 11:35:48

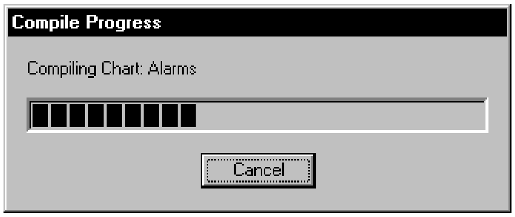
Warning: If you choose to continue, the strategy that is currently resident in the control engine will be REPLACED by this strategy.

Continue download?

   This message tells you that the strategy to be downloaded doesn't match the strategy already loaded on the control engine, either because it is a different strategy or because it has been changed since the last download.

3. To continue the download, click Yes.

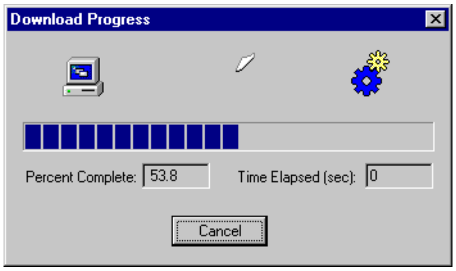
   As the strategy is compiled, the Compile Progress dialog box appears (usually very briefly).



Compile Progress

Compiling Chart: Alarms

If no errors occur, the Download Progress dialog box appears.



Percent Complete: 53.8

Time Elapsed (sec): 0

When the download is complete, you are in Debug mode. (If you receive errors, see Appendix A, "ioControl Troubleshooting.")

Compiling without Downloading

Sometimes you may want to compile without downloading, just to see if a chart, subroutine, or Strategy compiles correctly. You can compile the active chart or subroutine only, just the changes you have made to the strategy, or the entire strategy.

Compiling the Active Chart or Subroutine

Whenever a chart or subroutine window is open and active, you can compile just that chart or subroutine. To do so, in Configure mode, click the Compile Active View button on the toolbar, or choose Compile→Compile Chart. The menu option shows the name of the chart or subroutine you are compiling.

As soon as you choose the menu option, the chart or subroutine is saved and compiled. You are alerted only if errors are found.

Compiling Changes Only

To compile just the changes since the strategy was last compiled, in Configure mode, click the Compile Changes button on the toolbar, or choose Compile→Compile Changes. The menu option shows the name of the strategy you are compiling.

As soon as you choose the menu option, the strategy and all modified charts and subroutines are saved and compiled. You are alerted only if errors are found.

Compiling the Entire Strategy

To compile the entire strategy including all charts and subroutines, in Configure mode, click the Compile All button on the toolbar, or choose Compile→Compile All. The menu option shows the name of the strategy you are compiling.

As soon as you choose the menu option, the entire strategy is saved and compiled. The Compile Progress dialog box appears. You are alerted if errors are found.

Downloading Only

If your Strategy has been compiled, you can download it again quickly. Downloading again is useful if you want to run your strategy from a "clean slate" by reinitializing any variables that are set only on a strategy download.

To download a strategy that has already been compiled, you must be in Debug mode. Choose Control Engine→Download Strategy.

The Download Progress dialog box appears and your strategy is downloaded.

Downloading Without Using ioControl

If you are creating strategies for users who do not have ioControl on their systems (for example, if you are an integrator or OEM), you can make a control engine download file that can be downloaded to a SNAP Ultimate I/O brain or suitable SNAP controller using just ioTerminal or a DOS batch file. This one download file is built for a specific control engine but can also be downloaded to other similar control engines. It contains everything ioControl would download, including .per, .inc, and .crn files, control engine-specific files, and initialization information.

In most cases you will want the downloaded strategy to be saved to flash memory and to start automatically (autorun) when power is cycled to the control engine. Before you create the download file, follow the steps in "Saving to Flash on Every Download" on page 7-5. Check the boxes to have the strategy saved to flash memory after download and to set the autorun flag after download. This information will become part of the download file.

Creating the Control Engine Download (.cdf) File

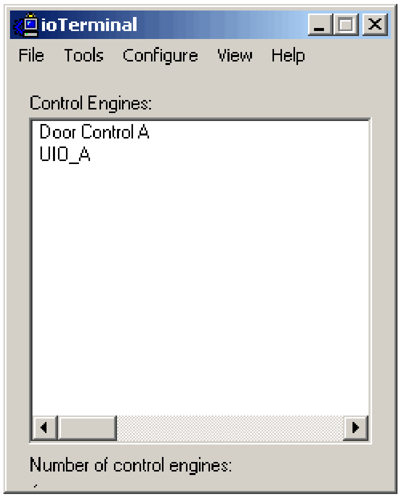
With the Strategy open in ioControl in Configure mode, right-click the name of the control engine in the Strategy Tree and choose Compile Control Engine Download File from the pop-up menu. (You can also choose Compile→Compile Control Engine Download File.)

The file is created in the same folder as the strategy, with a .cdf extension and a filename consisting of the strategy's name and the control engine's name (for example, MyStrategy.MyEngine.cdf).

Once the control engine download file is created, it can be downloaded using either ioTerminal or a DOS batch file you create.

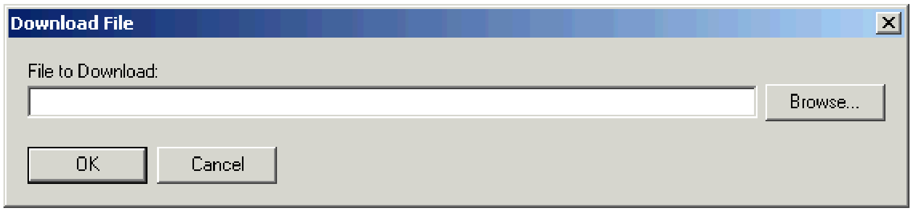
Downloading the .cdf File using ioTerminal

1. Click the Windows Start menu and choose Programs➞Opto22➞ioProject Software➞Tools➞ioTerminal.



2. Right-click the name of the control engine you want to download the file to.

3. In the pop-up menu, choose Download. In the submenu, choose Control Engine Download File.



4. Enter the path and filename of the .cdf file, or click the Browse button and navigate to it. When the filename appears in the File to Download field, click OK.

   The file is downloaded to the control engine, and a dialog box shows its progress.

Downloading the .cdf File Using a DOS Batch File

If you do not want your end user to have to use ioTerminal, you can create a DOS batch file to launch ioTerminal in the background and download the .cdf file. In addition to downloading the .cdf file, the batch file can also run or stop the strategy or even define the control engine on the PC that will download the file. ioTerminal must be installed on the PC where the batch file is used.

The following table lists actions you may want to take within ioTerminal:

To do this	Use this
Add a control engine	-a or -addce
Download the specified file to the specified control engine	-d or -download
Run the strategy in the specified control engine	-r or -run
Stop the strategy in the specified control engine	-s or -stop
Show help information for this function in ioTerminal	-h or -help
Start ioTerminal normally	<no arguments>

Format for lines in the batch file is as follows:

```
ioTerm [control_engine_name [-d filename] [-r]]
ioTerm -a control_engine_name tcp IP-address port retries timeout_ms
ioTerm -h
```

This example shows lines included in a batch file that will define the control engine, download the .cdf file to it, and then run the strategy:

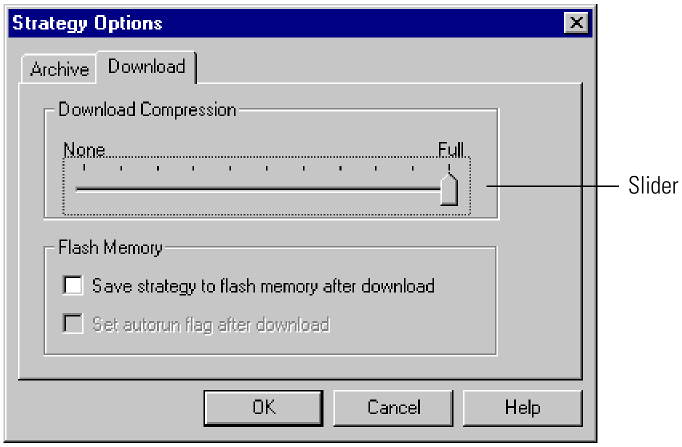
```
ioTerm -a MyCE tcp 10.20.30.40 22001 0 2000
ioTerm MyCE -d “c:\My_Project\MyStrategy.MyCE.CDF”
ioTerm MyCE -r
```

Changing Download Compression

If you have a very large Strategy, short timeouts, and slow connections on your network, you may need to decrease the compression level to download the strategy successfully. When you decrease compression, the strategy takes longer to download because it is sent in smaller chunks. If you are having difficulty downloading your strategy, follow these steps to decrease compression:

1. With the strategy open in Configure mode, choose File→Strategy Options.

2. In the Strategy Options dialog box, click the Download tab.



3. Move the slider to the left to reduce compression, and then click OK.

   You may need to experiment with the setting until the strategy downloads successfully.

Running a Strategy Manually

1. With the strategy open, choose Mode➞Debug.
2. Click the Run Strategy button ▶ (or press F5, or select Debug➞Run).

   You can also run a strategy from the Inspecting dialog box. See page 5-12.

Running a Strategy Automatically (Autorun)

You can set the Strategy to run automatically (autorun) if the control engine loses power and then restarts. In traditional Opto 22 controllers, the OptoControl autorun function was controlled by a jumper. For ioControl, it's controlled by the autorun flag. If the autorun flag is not set, the strategy must be started manually after power is restored to the control engine. The strategy must be saved in flash memory for autorun to work.

You can set the autorun flag in two ways:

- In Configure mode, save the strategy to flash memory and set the autorun flag every time the strategy is downloaded. (Be careful you do not save to flash too often, as flash memory eventually wears out.) See "Saving to Flash on Every Download" on page 7-5.
- In Debug mode, save the strategy to flash memory by choosing Control Engine➞Save Strategy to Flash. Then right-click the control engine in the Strategy Tree and choose Inspect from the pop-up menu. In the Inspect dialog box, enable Autorun. (See "Inspecting Control Engines and the Queue" on page 5-12 for more on the Inspect dialog box.)

Protecting a Running Strategy

If you want a Strategy to run automatically without interruption, you can protect the strategy by disabling all host communications to the control engine.

To protect the strategy, first make sure the strategy is saved to flash and that the autorun flag is set. To disable host communication, open ioManager and set the Control Engine port to 0, and then save that change to flash. For more information on setting this port, see form #1440, the *ioManager User's Guide*.

NOTE: This action also disables communication between ioDisplay and the control engine.

Stopping a Strategy

To stop the strategy, click the Stop Strategy button ■ (or press F3, or select Debug→Stop). You can also stop a strategy from the Inspecting dialog box; see page 5-12.

Debugging

Once the Strategy is running, if it doesn't appear to be working correctly, you can use several tools in Debug mode to figure out what the problem is. You can pause a chart or subroutine; step into, over, or out of each block; watch it slowly step through the blocks; or add a breakpoint to a block to stop the strategy just before it executes that block.

The chart's or subroutine's status is shown in the lower left-hand corner of its window. This corner shows whether the chart or subroutine is running, stopped, or suspended, and whether the debugging tools, such as stepping and breakpoints, are in effect. The chart or subroutine must be running in order to use these tools.

Choosing Debug Level

You can choose one of two levels of debugging:

- **Minimal Debug** lets you step from block to block, but does not allow you to step into blocks. Less information is downloaded to the control engine for minimal debugging, so downloading the strategy takes less time and less control engine memory. The strategy also runs slightly faster.
- **Full Debug** lets you step into blocks, so you can step through each instruction in an Action or Condition Block and through each line of OptoScript code in an OptoScript Block. If you are using OptoScript, you will probably want to spend the additional time to download your strategy at the full debug level.

To change debug level, make sure you are in Configure mode. From the Configure menu, choose Minimal Debug or Full Debug. The next time you enter Debug mode, the strategy will be compiled and downloaded with the new level.

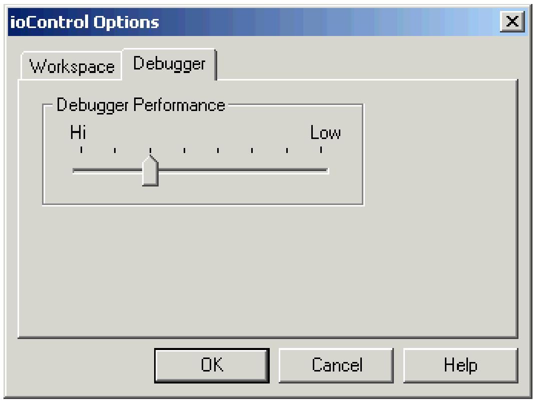
Changing Debugger Speed

Before you enter Debug mode, you may want to consider changing debugger speed. Depending on the number of charts and windows open in ioControl, and depending on other processing your computer is doing at the same time, you may find that running the debugger affects the computer's or the control engine's performance of other tasks. If necessary, you can slow down the debugger by increasing the time delay between debugging calls to the control engine, therefore leaving more processing time for other tasks.

In addition, a slower setting may be useful when checking communication using ioMessageViewer (see page A-7).

To change debugger speed, follow these steps:

1. With the strategy in Configure mode, choose Configure➞Options.
2. In the ioControl Options dialog box, click the Debugger tab.



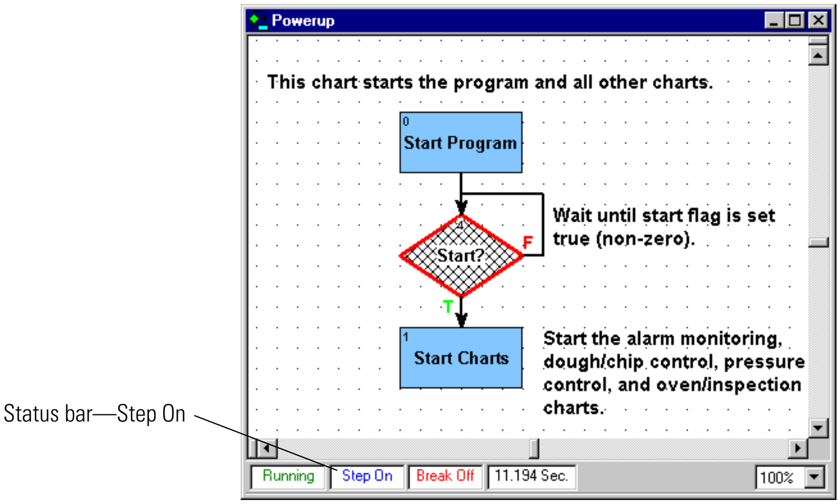
3. Click and drag the slider to the speed you want.

   The default speed is shown in the figure above. Since performance varies depending on your hardware and software, you may need to experiment to find the most efficient speed.

Pausing a Chart or Subroutine

You can temporarily stop any running chart or subroutine by pausing it. When you pause a chart or subroutine, it finishes the instruction it was executing, then stops at the next block (in Minimal Debug) or the next line (in Full Debug).

To pause the chart or subroutine in the active window, click the Pause Chart or Pause Subroutine button ⏸, or press F7, or select Debug➞Pause Chart or Debug➞Pause Subroutine. Here's an example of a paused chart:



This chart starts the program and all other charts.

Wait until start flag is set true (non-zero).

Start the alarm monitoring, dough/chip control, pressure control, and oven/inspection charts.

Hatch marks and a red outline appear on the Start block, indicating that this block is to be executed next. The status bar shows Step On, which means you can step through the chart or subroutine if you wish.

Stepping Through a Chart or Subroutine

When you step through a chart or subroutine, you control the timing and execution of its commands in a running Strategy. You can see what commands are being executed when, and you can monitor the status of variables and I/O that are affected by the commands.

There are two types of stepping: single-stepping and automatic stepping. Use single stepping to go through flowchart blocks at your own pace. Use auto stepping to watch the flowchart step automatically.

CAUTION: *Since stepping through a running chart or subroutine—even auto stepping—slows down execution, be cautious if your strategy is running on real equipment. For example, stepping through a strategy might leave a valve open much longer than it should be.*

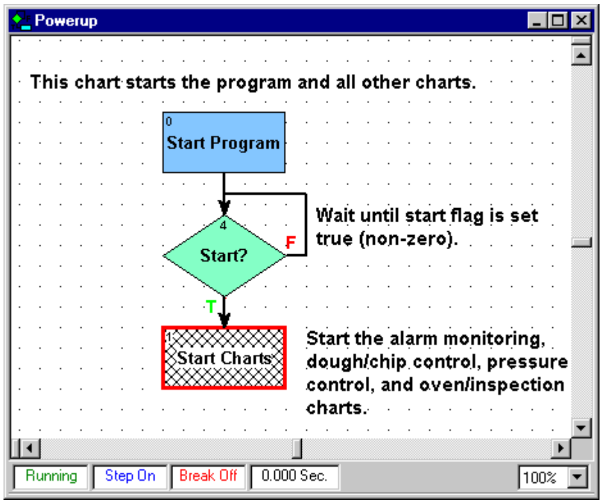
Single Stepping

When you are debugging a Strategy, start by using the Step Over button to go through a chart one block at a time. The Step Over button may be all you need to find any problems. If necessary, go back to Configure mode and change to full debug (see "Choosing Debug Level" on page 7-13) so you can step into blocks and execute one line at a time.

1. Pause the chart or subroutine to be stepped through by pressing the Pause Chart or Pause Subroutine button ⏸.

2. To step to the next command block, click the Step Over button [|]↓ (or press F10, or select Debug→Step Over).

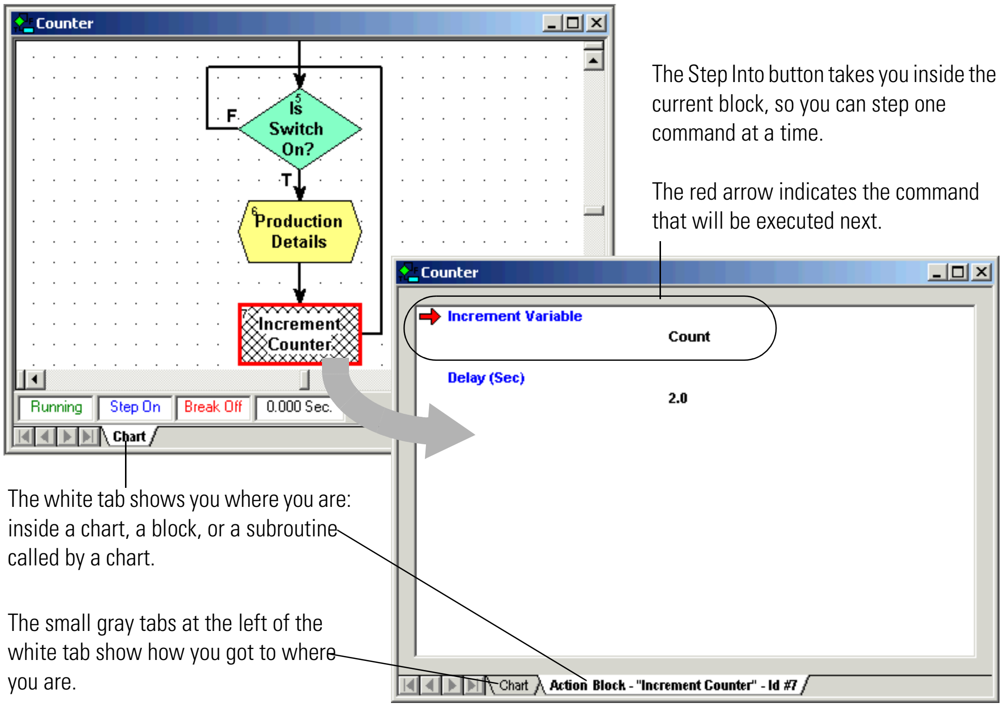
   The commands in the highlighted block are executed, the hatch marks move to the next command block, and the chart pauses again. Compare the chart below to the one on page 7-15. The hatch mark has moved to the next block.


Powerup
This chart starts the program and all other charts.
0
Start Program
4
Start?
F
T
Wait until start flag is set true (non-zero).
1
Start Charts
Start the alarm monitoring, dough/chip control, pressure control, and oven/inspection charts.
Running
Step On
Break Off
0.000 Sec.
100%

3. If you need to step inside flowchart blocks and move through them one command at a time (or in OptoScript Blocks, one line of code at a time), make sure you have downloaded your strategy at the full debug level. See "Choosing Debug Level" on page 7-13 for help.

4. If the chart or subroutine is not already paused, press the Pause Chart or Pause Subroutine button ▮▮ .

5. To step inside the block you are on (the one with the hatch marks), click the Step Into button (or press F11, or select Debug➞Step Into).



Increment Variable
Count

Delay (Sec)
2.0

The Step Into button takes you inside the current block, so you can step one command at a time.

The red arrow indicates the command that will be executed next.

The white tab shows where you are: inside a chart, a block, or a subroutine called by a chart.

The small gray tabs at the left of the white tab show how you got to where you are.

6. To move from line to line, click either the Step Into or the Step Over button.

   If a command within the block calls a subroutine, Step Into takes you into the subroutine. Step Over skips over the subroutine.

7. To step from a command inside a block and go to the next block, click Step Out .

   Clicking Step Out when you are on a block, rather than inside it, unpauses the chart. In a subroutine, clicking Step Out takes you out of the subroutine.

Auto Stepping

Autostepping lets you watch a chart's or a subroutine's logic in slow motion, one block at a time. A chart or subroutine does not have to be paused before auto stepping can begin. To begin auto stepping, click the Auto Step button , press F8, or select Debug➞Auto Step Chart.

Step Auto appears in the status bar. The hatch marks move from block to block as each block's commands are executed. When you reach a block whose code is currently being executed, the highlight around the block becomes changing shades of green instead of solid red (unless the block is executed very quickly).

A chart that contains flow-through logic stops when it has been stepped through. In a chart that contains loop logic, the autostepping continues until you stop it by pressing the Auto Step button again.

Setting and Removing Breakpoints

Sometimes you want to see the action at one or two blocks without having to step through an entire chart or subroutine. You can use a breakpoint to stop a running chart or subroutine just before a block is executed.

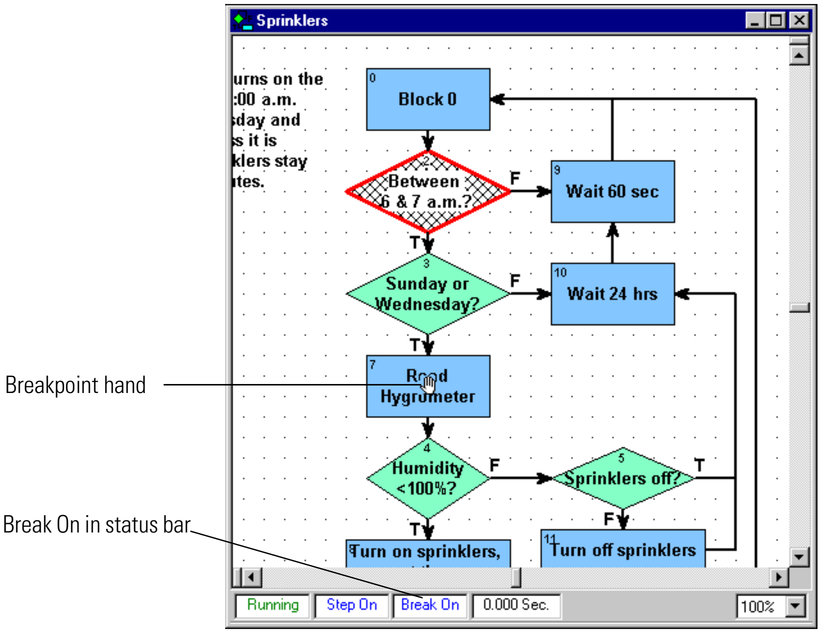
You can set a breakpoint at any block in any chart or subroutine, whether it is running or stopped, paused or auto stepped. The strategy does not need to be running. You can set up to 16 breakpoints in one chart or subroutine. However, you cannot set a breakpoint inside a block.

To set a breakpoint, follow these steps:

1. With the chart or subroutine open and in Debug mode, click the Breakpoint Tool button ✋.

   The pointer turns into a hand.

2. Click the target block to mark it with the breakpoint hand.



   The breakpoint hand appears on the block and Break On appears in the status bar.

3. Click other blocks to set additional breakpoints, or click blocks currently marked with a hand to remove the breakpoint.

4. When you have finished marking or removing breakpoints, click the Breakpoint button again or click the right mouse button within the window.

   When the chart or subroutine runs, it pauses just before executing the breakpoint block. You can inspect variables or I/O points, disable strategy elements, change values, and so on to see the effect the block has.

5. To single step past the breakpoint, click the Step Block or Step Line button. Or to run the chart or subroutine at full speed after the breakpoint, click the Pause Chart button.

Managing Multiple Breakpoints

You can quickly set multiple breakpoints in several charts and subroutines, or you can see all the breakpoints you have set at once.

1. Press CTRL + B or select Debug→Breakpoints.

   The Breakpoints dialog box appears, showing all the breakpoints set in the strategy.

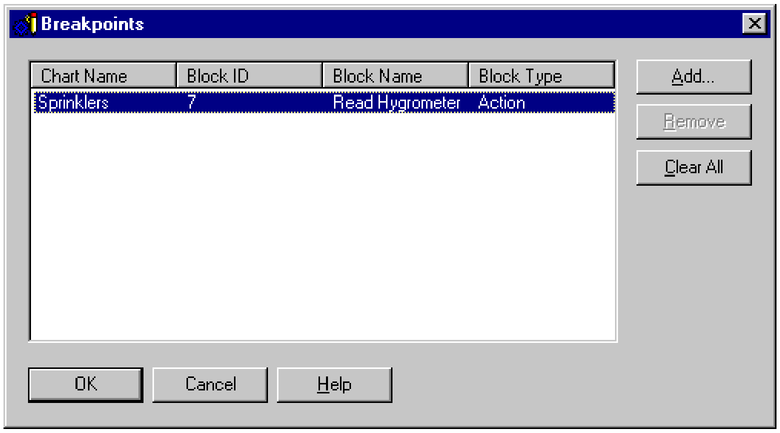


Chart Name	Block ID	Block Name	Block Type
Sprinklers	7	Read Hygrometer	Action

2. To add new breakpoints, click Add.

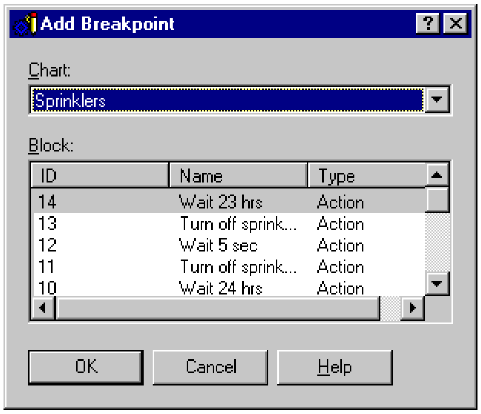


Chart: Sprinklers

Block:

ID	Name	Type
14	Wait 23 hrs	Action
13	Turn off sprink...	Action
12	Wait 5 sec	Action
11	Turn off sprink...	Action
10	Wait 24 hrs	Action

3. From the Chart drop-down list, select the chart in this strategy in which you want to place a breakpoint.

   Every block in that chart appears in the Block list. You can click the ID, Name, or Type column labels to sort the blocks numerically by ID or alphabetically by name or type.

4. To add a breakpoint, highlight a block and click OK.

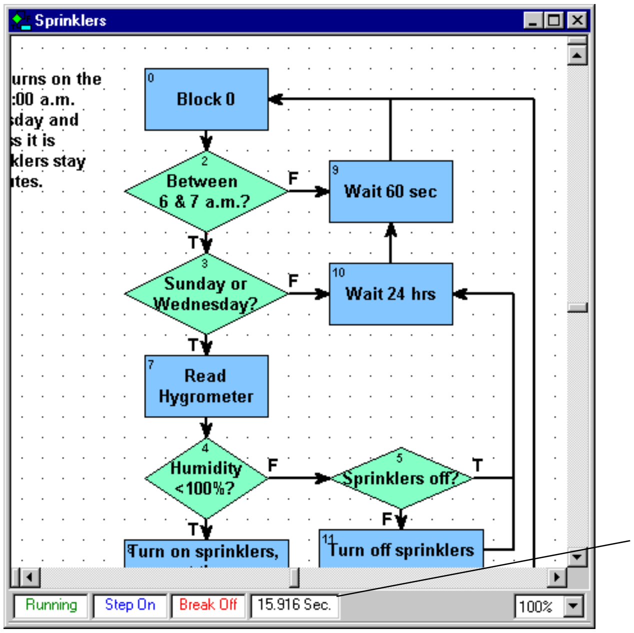
   The new breakpoint appears in the Breakpoints dialog box.

5. To delete a breakpoint, highlight it and click Remove. To delete all breakpoints in the strategy at once, click Clear All.

6. When you have finished making changes, click OK.

Interpreting Elapsed Times

As you debug your Strategy, you may notice elapsed time readings appearing in a chart's or subroutine's status bar, as shown below.



Elapsed time

Elapsed time readings can help you determine how much time a chart, a subroutine, or a single block takes to execute. The readings have slightly different meanings depending on what you did to make them appear, as described in the table below:

When you . . .	Elapsed time represents . . .
Run a chart or subroutine and pause it	Time since the chart or subroutine started or was last paused
Single step (by line or block)	Time to execute the previous block
Auto step	Time to execute the most recently executed block
Hit a breakpoint	Time since the last pause, or if the chart or subroutine was not paused, elapsed time since it started running

Viewing and Printing

You can view and print several helpful things in a Strategy as described in the following topics:

- "Viewing Strategy Filename and Path" (below)
- "Viewing an Individual Chart or Subroutine" on page 7-21
- "Viewing All Charts in a Strategy" on page 7-21
- "Printing Chart or Subroutine Graphics" on page 7-23
- "Viewing and Printing Strategy or Subroutine Commands" on page 7-25
- "Viewing and Printing Strategy or Subroutine Elements" on page 7-26
- "Viewing and Printing a Cross Reference" on page 7-28
- "View and Print a Bill of Materials" on page 7-29

For information on viewing and changing I/O units, see "Inspecting I/O in Debug Mode" on page 6-51. For variables, see "Viewing Variables in Debug Mode" on page 9-14.

Viewing Strategy Filename and Path

To see an open Strategy's filename and path, choose File→Strategy Information. A dialog box appears showing the path and filename.

Viewing an Individual Chart or Subroutine

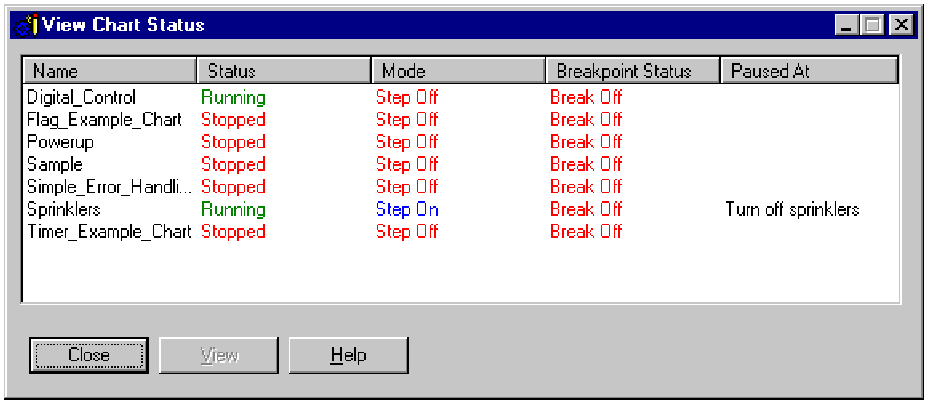
To view an individual chart or subroutine, double-click its name on the Strategy Tree, or choose Chart→Open or Subroutine→Open. You can open as many of these windows as you need. The names of open windows appear on tabs at the bottom of the ioControl main window. Click a tab to bring its window into view.

Viewing All Charts in a Strategy

You can see the status of all charts at once and change a chart's status without having to open it.

1. Make sure the strategy is open and in Debug mode. On the Strategy Tree, double-click the Charts folder.

The View Chart Status dialog box appears, showing every chart in the strategy:



Name	Status	Mode	Breakpoint Status	Paused At
Digital_Control	Running	Step Off	Break Off	
Flag_Example_Chart	Stopped	Step Off	Break Off	
Powerup	Stopped	Step Off	Break Off	
Sample	Stopped	Step Off	Break Off	
Simple_Error_Handli...	Stopped	Step Off	Break Off	
Sprinklers	Running	Step On	Break Off	Turn off sprinklers
Timer_Example_Chart	Stopped	Step Off	Break Off	

2. To change the status of a chart, double-click the chart name.

   The View Chart dialog box appears, showing the chart name, chart status, run mode, and breakpoint status. If the chart is paused (mode is Step On), the block at which it is paused is shown in the Paused At field. In the figure below, the chart is not paused:

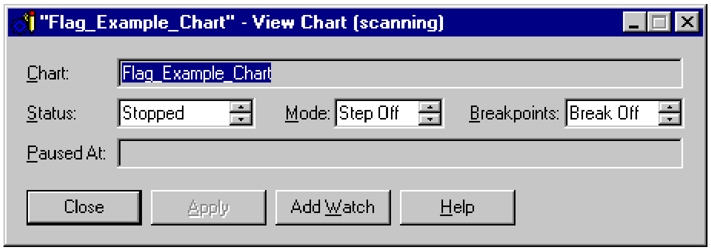


Chart: Flag_Example_Chart

Status: Stopped Mode: Step Off Breakpoints: Break Off

Paused At:

   The title bar shows whether scanning is occurring. Scanning stops when you click one of the changeable fields (Status, Mode, and Breakpoints) and resumes once you click Apply, another button, or one of the other fields. If scanning resumes before you click Apply, any changes you made are lost.

3. To stop, run, or suspend a chart, click an arrow in the Status field to select the option. Click Apply.

4. To turn pausing on or off, click an arrow in the Mode field to select Step On or Step Off. Click Apply.

5. To observe or ignore any breakpoints set in the chart, click an arrow in the Breakpoints field to select Break On or Break Off. Click Apply.

   This action does not clear or set breakpoints, but just determines whether the chart stops at breakpoints when it is running.

   Chart changes occur as soon as you click Apply.

6. To add the chart to a watch window so you can monitor it with other strategy elements, click Add Watch. In the dialog box, choose what to watch. Select an existing watch window to add this chart to, or create a new watch window.

See "Using Watch Windows for Monitoring" on page 6-73 for more information on watch windows.

7. When you have finished making changes, click Close to return to the View Chart Status dialog box.

Printing Chart or Subroutine Graphics

You can print a chart or subroutine just as it appears on screen. You can also print all charts within a Strategy. When printing a single chart or subroutine, you can preview the image to make sure it's what you want before you print it.

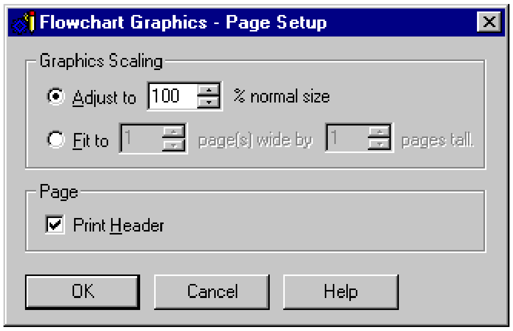
NOTE: If you have trouble printing graphics, set your printer for PostScript emulation.

Setting Up the Page

Before printing graphics, you should verify your page setup, which determines how each graphic appears on a page.

1. In Configure mode, select File→Page Setup.

   The Page Setup dialog box appears.



Flowchart Graphics - Page Setup

Graphics Scaling

● Adjust to 100 % normal size

○ Fit to 1 page(s) wide by 1 pages tall.

Page

☑ Print Header

2. In the Graphics Scaling area, choose whether you want each flowchart to print at a fixed percentage of normal size or to span a specific number of pages.

   - To print at a fixed percentage, click the Adjust To option and specify any scaling from one percent to 1,000 percent.

     You can type in a number or click the arrows to go up or down to the next increment of 25 percent. Typically, percentages between 50 percent and 200 percent work the best.

   - To print to a specific number of pages, click the Fit To option and select the number of pages wide and tall you would like each chart to print.

     If you choose one for each dimension, each chart prints to a single page. For each dimension, you can specify any integer between one and 255, but be careful. Selecting values of five and five, for example, would cause each chart to print five pages wide and five pages long, a total of 25 pages.

3. (Recommended) To print a header on each page, put a check mark in the Print Header box.

   The header lists the strategy name, chart or subroutine name, date and time of printing, page number, and column and row of the page with respect to the full chart printout.

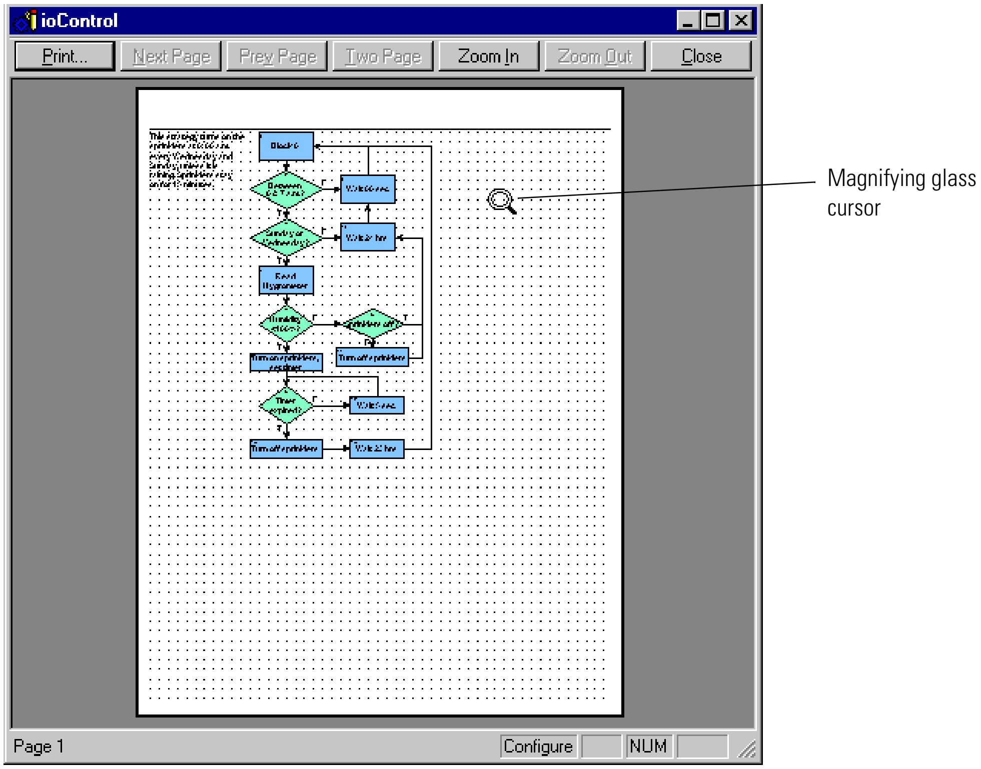
4. Click OK to save your settings.

Previewing a Flowchart Printout

1. To see how a chart or subroutine will print before actually printing it, open the chart or subroutine window.

2. From the Chart or Subroutine menu, select Print Preview Graphics.

   The preview window appears, showing the image as it will print. The cursor becomes a magnifying glass.



3. To zoom in at 200 percent, click where you want to see more closely. Click again to zoom in at 400 percent. Click a third time to return to 100 percent view.

   You can also use the Zoom In and Zoom Out buttons at the top of the window to zoom in or out with respect to the top left corner of the image.

4. If the image spans more than one page, click the Next Page or Prev Page buttons to view the next or previous page. To switch between a single-page view and a double-page view, click the Two Page/One Page button.

5. To print, click the Print button to open the standard Windows Print dialog box. To change settings before printing, click Close and see "Setting Up the Page" on page 7-23.

Printing One Chart or Subroutine

1. To print one chart or subroutine, open its window.

2. From the Chart or Subroutine menu, select Print Graphics.

3. In the standard Windows Print dialog box, do one of the following:

   - To print to a printer, select the printer, page range, and number of copies. Click OK.
   - To print to a file, select Print to file and click OK. In the dialog box, enter the file name and location.

   Your chart or subroutine is printed.

Printing All Charts in a Strategy

***CAUTION:** You can print all charts included in a strategy, but be sure that's what you want to do before you begin. You **cannot cancel** once printing has started.*

1. To print all charts within a strategy, open the strategy and check the page setup.

   For help, see "Setting Up the Page" on page 7-23.

2. Select File→Print All Graphics.

   Printing begins immediately; no Print dialog box appears. Messages inform you of each chart's printing progress. To skip printing a particular chart, click Cancel when its message appears.

Viewing and Printing Strategy or Subroutine Commands

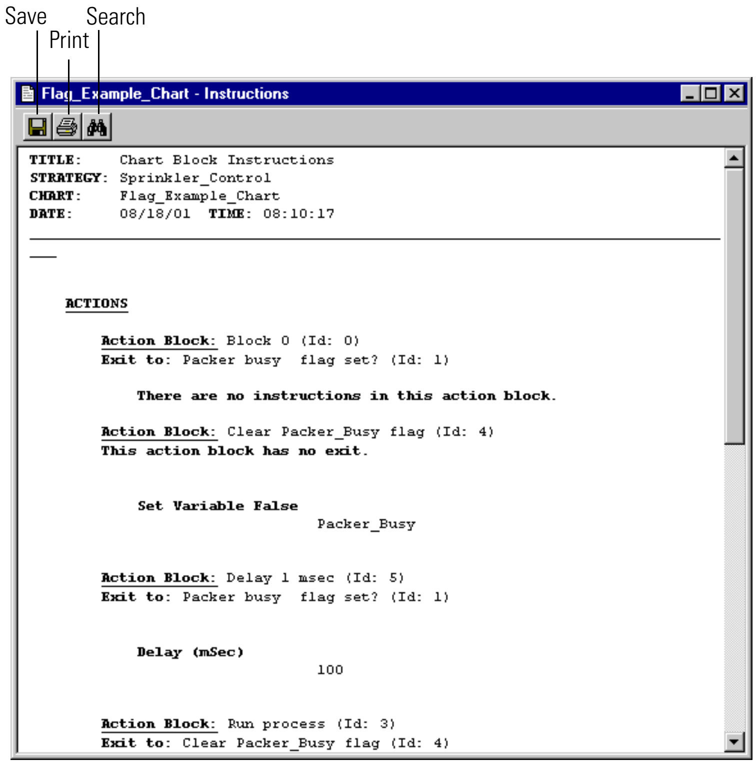
You must be in Configure mode to view and print commands.

1. To view all commands (instructions) in a chart or subroutine, open its window and select View/Print Instructions from the Chart or Subroutine menu. Choose whether to sort instructions by block name or block ID number.

2. To view all instructions in an entire strategy, select File→View/Print→All Chart Instructions.

   NOTE: Subroutine instructions are not included; you can print them separately.

ioControl processes the information and displays it in the Instructions window.



```
TITLE:    Chart Block Instructions
STRATEGY: Sprinkler_Control
CHART:    Flag_Example_Chart
DATE:     08/18/01  TIME: 08:10:17
____________________________________________________________________________
___


    ACTIONS

        Action Block: Block 0 (Id: 0)
        Exit to: Packer busy  flag set? (Id: 1)

            There are no instructions in this action block.

        Action Block: Clear Packer_Busy flag (Id: 4)
        This action block has no exit.


            Set Variable False
                                Packer_Busy


        Action Block: Delay 1 msec (Id: 5)
        Exit to: Packer busy  flag set? (Id: 1)


            Delay (mSec)
                                100


        Action Block: Run process (Id: 3)
        Exit to: Clear Packer_Busy flag (Id: 4)
```

You may need to resize the window and use the scroll bar to see all the data. Blocks and their instructions are listed in alphabetical or ID number order by type of block: Action Blocks first, then OptoScript Blocks, then Condition Blocks, and finally Continue Blocks.

3. To print the data, click the print button on the toolbar. To save it to a text file, click the save button. To search the data, click the search button. When finished, close the window.

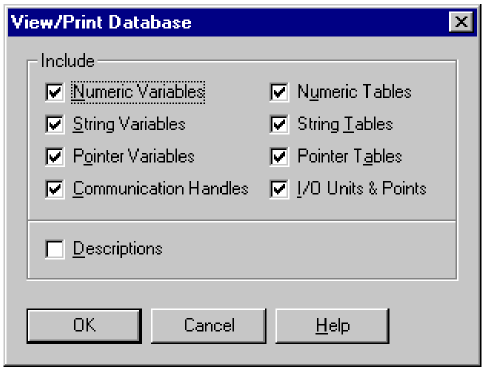
Viewing and Printing Strategy or Subroutine Elements

You must be in Configure mode.

1. To view a summary of I/O elements and variables configured in a strategy, select File→View/Print→Database.

2. To view the same summary for a subroutine, open the subroutine window and select Subroutine→View/Print→Database.

The View/Print Database dialog box appears.



View/Print Database

Include

☑ Numeric Variables ☑ Numeric Tables
☑ String Variables ☑ String Tables
☑ Pointer Variables ☑ Pointer Tables
☑ Communication Handles ☑ I/O Units & Points

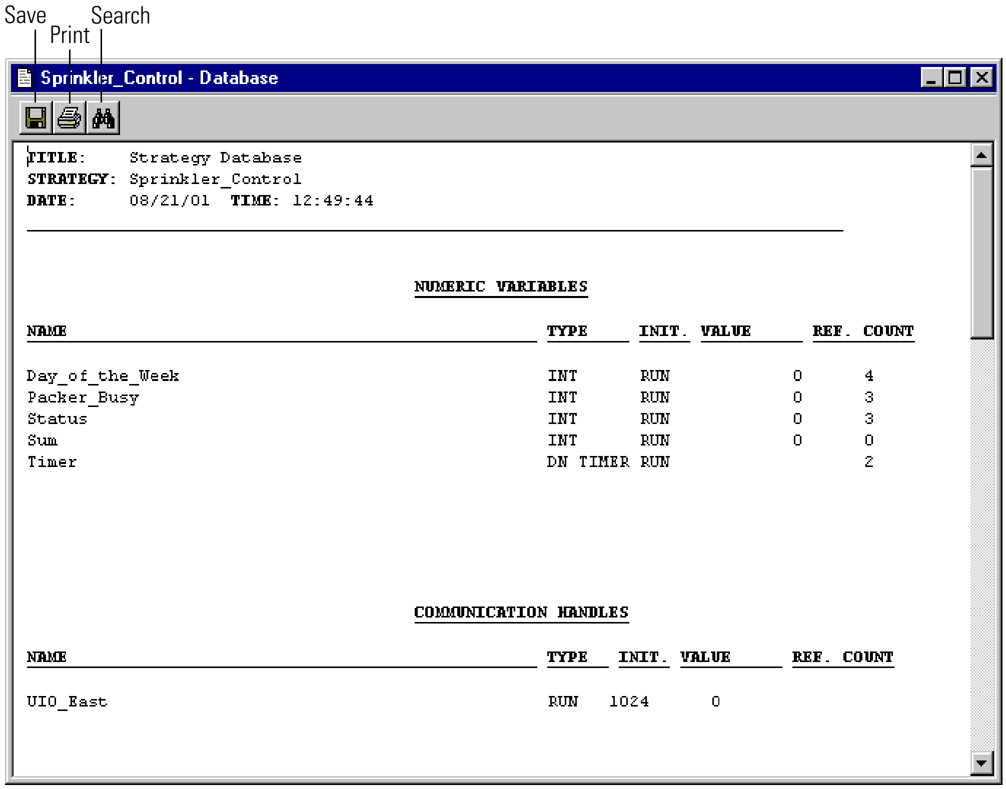
☐ Descriptions

3. Make sure all element types you want to include are checked. Click to uncheck any elements you do not want.

4. To include descriptive comments associated with the elements, click to put a check mark next to Descriptions.

5. Click OK.

   ioControl processes the database and puts the data in the Database window.



```
TITLE:    Strategy Database
STRATEGY: Sprinkler_Control
DATE:     08/21/01  TIME: 12:49:44

                         NUMERIC VARIABLES

NAME                                              TYPE     INIT. VALUE     REF. COUNT

Day_of_the_Week                                   INT      RUN         0       4
Packer_Busy                                       INT      RUN         0       3
Status                                            INT      RUN         0       3
Sum                                               INT      RUN         0       0
Timer                                             DN TIMER RUN                 2

                         COMMUNICATION HANDLES

NAME                                              TYPE   INIT. VALUE     REF. COUNT

UIO_East                                          RUN    1024      0
```

You may need to resize the window and use the scroll bar to see all the data. For each element the name and reference count (that is, how many times the element is used in strategy commands) are shown, plus other information depending on the element type. The figure above shows numeric variables and communication handles.

6. To print the data, click the print button on the toolbar. To save it to a text file, click the save button. To search the data, click the search button. When finished, close the window.

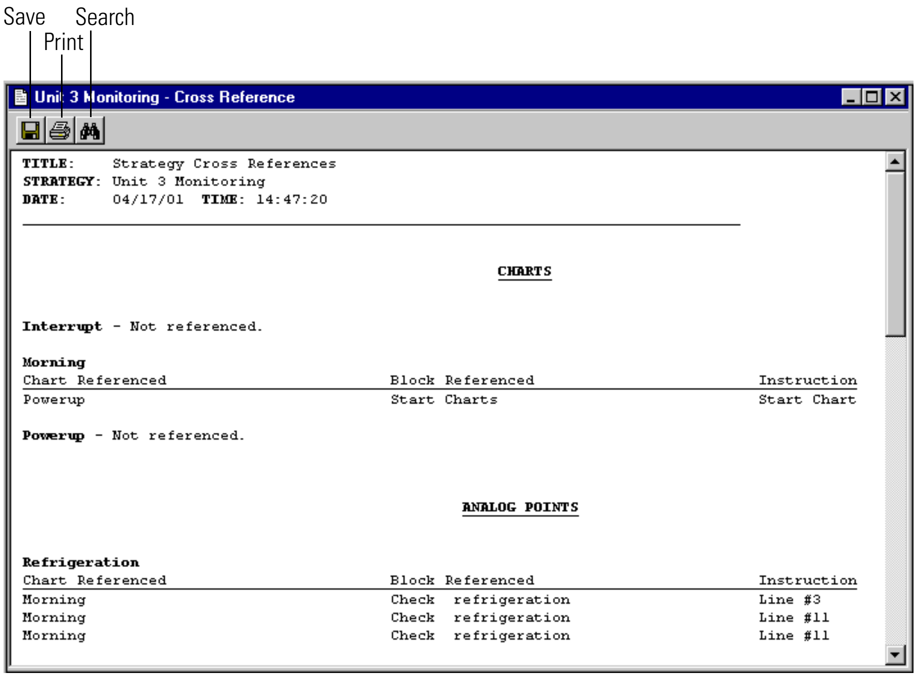
Viewing and Printing a Cross Reference

You can view and print a report of every operand in your Strategy or subroutine—charts, I/O units, Analog Points, Digital Points, communication handles, numeric variables, string variables, pointer variables, numeric tables, string tables, and pointer tables. The operands are cross-referenced to the charts, blocks, and instructions in which they are used.

1. To produce a cross reference for a strategy, open it and select File➞View/Print➞Cross Reference.

2. To view a similar report for a subroutine, open the subroutine window and select Subroutine➞View/Print➞Cross Reference.

   ioControl processes the data and puts it in the Cross Reference window.



```
TITLE:    Strategy Cross References
STRATEGY: Unit 3 Monitoring
DATE:     04/17/01  TIME: 14:47:20
_______________________________________________________________________

                                  CHARTS

Interrupt - Not referenced.

Morning
Chart Referenced                    Block Referenced                    Instruction
Powerup                             Start Charts                        Start Chart

Powerup - Not referenced.

                                  ANALOG POINTS

Refrigeration
Chart Referenced                    Block Referenced                    Instruction
Morning                             Check  refrigeration                Line #3
Morning                             Check  refrigeration                Line #11
Morning                             Check  refrigeration                Line #11
```

You may need to resize the window and use the scroll bar to see all the data. Notice that the Instruction column (at right) shows the line number the operand appears in when it is in OptoScript code.

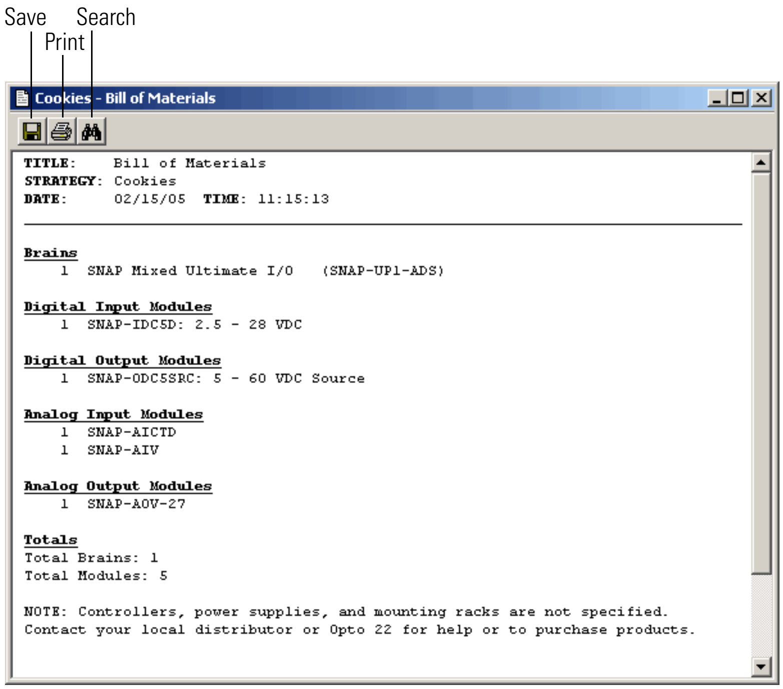
3. To print the data, click the print button on the toolbar. To save it to a text file, click the save button. To search the data, click the search button. When finished, close the window.

View and Print a Bill of Materials

You can view and print a bill of materials (BOM) that lists all the I/O units and I/O modules (analog and standard digital) required to run the Strategy. (Special-purpose modules, such as serial and high-density digital modules, are not included in the BOM.)

1. To produce a BOM for a strategy, open it and select File➞View/Print➞Bill of Materials.

   ioControl processes the data and puts it in the Bill of Materials window.



```
TITLE:    Bill of Materials
STRATEGY: Cookies
DATE:     02/15/05  TIME: 11:15:13

Brains
    1  SNAP Mixed Ultimate I/O   (SNAP-UP1-ADS)

Digital Input Modules
    1  SNAP-IDC5D: 2.5 - 28 VDC

Digital Output Modules
    1  SNAP-ODC5SRC: 5 - 60 VDC Source

Analog Input Modules
    1  SNAP-AICTD
    1  SNAP-AIV

Analog Output Modules
    1  SNAP-AOV-27

Totals
Total Brains: 1
Total Modules: 5

NOTE: Controllers, power supplies, and mounting racks are not specified.
Contact your local distributor or Opto 22 for help or to purchase products.
```

   You may need to resize the window and use the scroll bar to see all the data.

2. To print the data, click the print button on the toolbar. To save it to a text file, click the save button. To search the data, click the search button. When finished, close the window.

Searching and Replacing

You can search a chart, subroutine, or Strategy for missing connections, empty Condition Blocks, or any command or operand. An operand is anything that can be affected by a command, including charts, I/O units, Analog Points, Digital Points, and all kinds of variables. Searching includes OptoScript code within OptoScript Blocks.

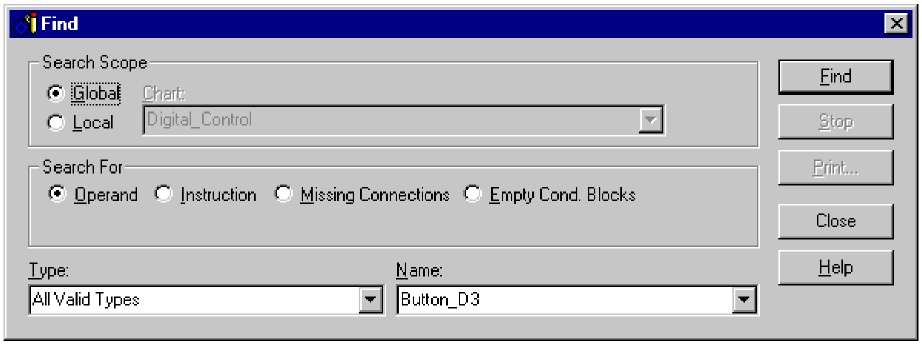
You can also replace instructions or operands with similar items.

Searching

You can search a a Strategy or one of its charts, or you can search a subroutine.

1. Open the strategy or subroutine and select Edit➞Find.

   The Find dialog box appears.



Find

Search Scope

● Global ○ Local

Chart: Digital_Control

Search For

● Operand ○ Instruction ○ Missing Connections ○ Empty Cond. Blocks

Type: All Valid Types

Name: Button_D3

2. Under Search Scope, to search the entire strategy, click Global. To search one chart only, click Local and choose the chart name from the drop-down list.

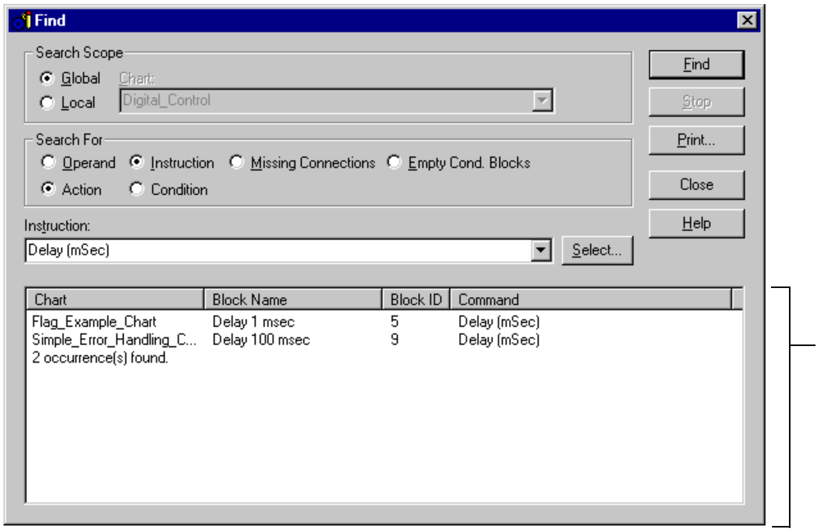
   If you are searching a subroutine, the search is Local and the subroutine’s name is shown.

3. Under Search For, choose one of the following:

   - To search for a chart, an I/O unit or point, or a variable, click Operand. In the Type and Name fields, choose the operand you want from the drop-down list.
   - To search for an instruction, click Instruction. Click Action or Condition, and choose the instruction you want from the drop-down list.
   - To search for blocks that are not connected to other blocks, click Missing Connections.
   - To search for Condition Blocks that have no instructions, click Empty Cond. Blocks.

4. Click Find.

The dialog box expands and the search results appear at the bottom.



Search Scope
● Global ○ Local
Chart: Digital_Control

Search For
○ Operand ● Instruction ○ Missing Connections ○ Empty Cond. Blocks
● Action ○ Condition

Instruction: Delay (mSec)

Chart	Block Name	Block ID	Command
Flag_Example_Chart	Delay 1 msec	5	Delay (mSec)
Simple_Error_Handling_C...	Delay 100 msec	9	Delay (mSec)
2 occurrence(s) found.			

Search results

For more information on any item in the search results, try double-clicking the item.

5. To save the search results to a file or to print them, click Print. In the window that opens, click the disk button to save or the printer button to print your search results.

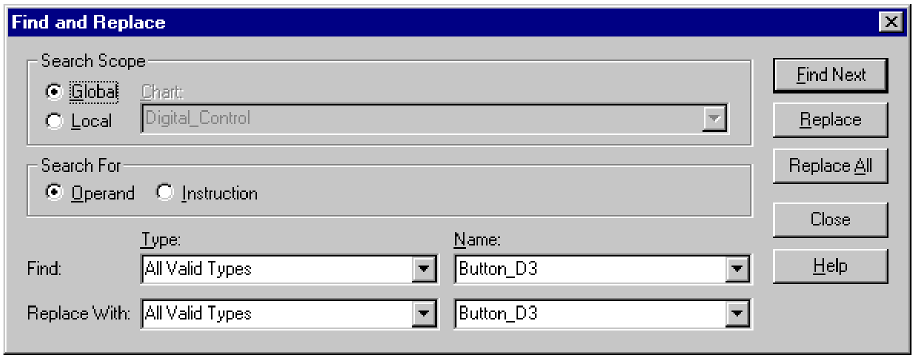
6. When you have finished your search, close the Find dialog box.

Replacing

You can also replace any operand or instruction with a similar item. As in searching, you can replace items in a Strategy or one of its charts, or you can replace items in a subroutine.

1. Open the strategy or subroutine, and select Edit➞Replace.

   The Find and Replace dialog box appears.



Search Scope
● Global ○ Local
Chart: Digital_Control

Search For
● Operand ○ Instruction

	Type:	Name:
Find:	All Valid Types	Button_D3
Replace With:	All Valid Types	Button_D3

2. Under Search Scope, to search the entire strategy, click Global. To search one chart only, click Local and choose the chart name from the drop-down list.

   If you are searching a subroutine, the search is Local and the subroutine’s name is shown.

3. Under Search For, choose one of the following:

- To search for a chart, an I/O unit or point, or a variable, click Operand. In the Find Type and Name fields, choose the operand you want to replace from the drop-down list. In the Replace With Type and Name fields, choose the operand you want to use instead.
- To search for an instruction, click Instruction. Click Action or Condition, and choose the instruction you want to replace from the Find drop-down list. In the Replace With drop-down list, choose the instruction you want to use instead.

4. Click Find Next.

When the first occurrence of the operand or instruction is found, the Instructions dialog box it appears in is displayed.

5. To replace this occurrence, click Replace. To skip it and find the next one, click Find Next.

If you are replacing operands, you can replace all occurrences at once by clicking Replace All. If you are replacing instructions, you must verify each one.

6. If the Edit Instructions dialog box appears, make any necessary changes and click OK to save them before moving on.

7. When replacements are finished, close the Find and Replace dialog box.

Working with Flowcharts

Introduction

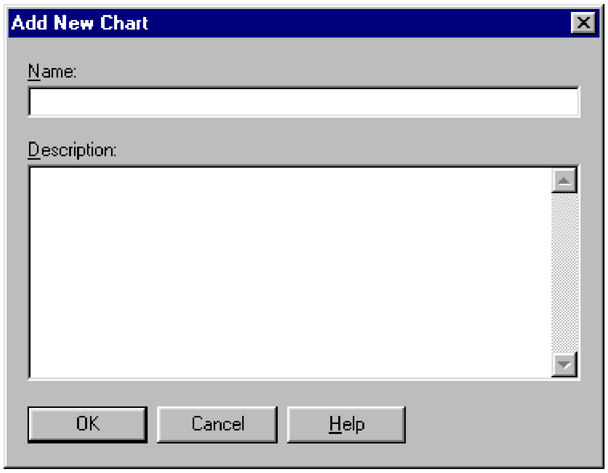
This chapter shows you how to work with flowcharts, the building blocks of your strategy. When you create a new strategy, one chart is created for you: the Powerup Chart. You must create all the other charts to do the work of the strategy.

In this Chapter

Creating a New Chart

1. With your strategy open and in Configure mode, select Chart→New, or right-click the Charts folder on the Strategy Tree and select New from the pop-up menu.

   The Add New Chart dialog box appears.



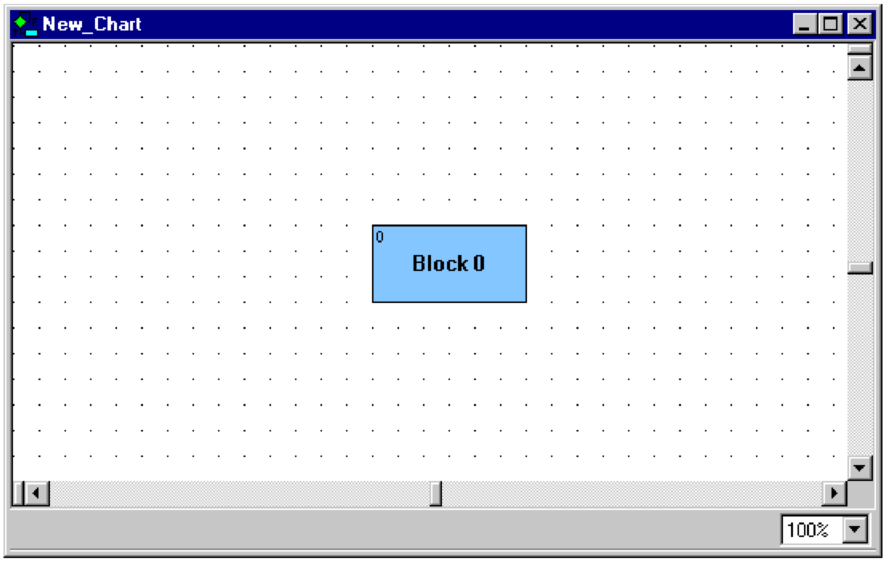
2. Enter a name for the new chart.

   The name must start with a letter, but may also include numbers and underscores. If you type spaces, they are converted to underscores. All other characters are ignored.

3. (Optional) Type a description.

4. Click OK.

   The new chart is listed on the Strategy Tree under the Charts folder, and the new chart window appears. Block 0, the starting block, is shown automatically. No matter how many other blocks you add or where you place them, block 0 is always the first block to be executed in the chart.



NOTE: Because chart windows show a small portion of a potentially large chart, a small movement with the scroll bar can mean a big change in what you see. If you lose your flowchart in the window, select View→Center on Block and choose the block you want to see in the middle of the screen.

For information on splitting and zooming chart windows, see page 3-18.

Working with Chart Elements

What's In a Chart?

Charts can contain four kinds of flowchart blocks, lines connecting the blocks, and text.

Action Blocks are rectangles that contain one or more commands (instructions) that do the work of the strategy, such as turning things on or off, setting variables, and so on. See Chapter

9, "Using Variables and Commands," for more information. Action blocks can have more than one entrance but only one exit.

Condition Blocks are diamonds containing questions that control the logical flow of a strategy. Condition blocks can have many entrances, but only two exits: True and False.

OptoScript Blocks are hexagons containing OptoScript code, a procedural language you may want to use to simplify certain tasks. See Chapter 11, "Using OptoScript," for more information. OptoScript blocks can have more than one entrance but only one exit.

Continue Blocks are ovals that contain no commands, but simply route chart logic to a new location, such as to the top of a chart. These blocks help keep charts neat by avoiding awkward connections between two blocks that are far apart.

Connections are lines with arrows that connect one block to the next, directing the flow of strategy logic.

Text explains the chart's purpose and elements for anyone who needs to understand them later.

Using the Drawing Toolbar

The drawing toolbar includes tools for each of the elements plus a Select tool, or pointer, for manipulating elements:

Changing the Appearance of Elements in a Chart Window

You can change the background appearance of charts or subroutines, the color and size of blocks and text, and the color of connection lines. Depending on the scope you want to affect, you can change these window properties at three levels:

- Across ioControl—to change the appearance of all new charts in all new strategies, and all new subroutines.
- Across a strategy—to change the appearance of all new charts in the open strategy.
- For the open chart or subroutine—to change the appearance of all new elements in the open chart or subroutine window.

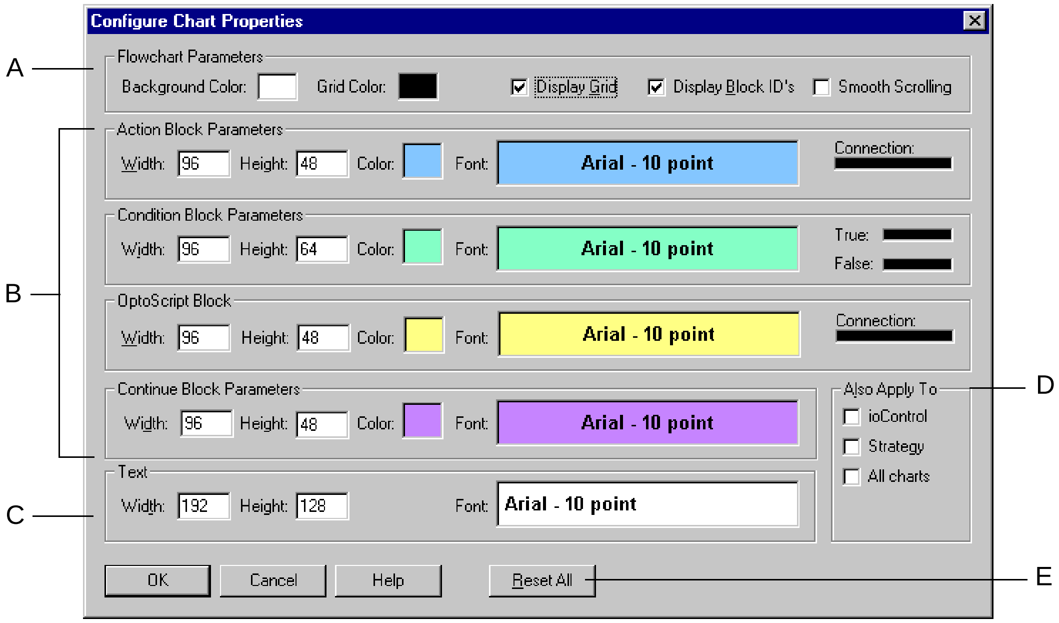
*IMPORTANT: Note that most changes affect only new charts and their elements. Existing charts, subroutines, and elements are **not** changed. To avoid having to go back and change each item individually, make sure you set the defaults the way you want them before you create new charts. Once you have changed the defaults, see page 8-5 to change existing elements to match the new defaults.*

To change the appearance of charts and elements, follow these steps:

1. Choose one of the following, depending on the scope you want to change:
   - To change all new charts in all new strategies and all new subroutines, choose Configure→Default Properties to open the Configure ioControl Default Properties dialog box.

- To change all new charts in the open strategy only, choose File➞Strategy Properties to open the Configure Strategy Properties dialog box.
- To change new elements in the open chart or subroutine only, choose Chart➞Properties or Subroutine➞Properties to open the Configure Chart Properties or Configure Subroutine Properties dialog box.

The dialog box that opens looks like this, except that its title may be different.



Configure Chart Properties

Flowchart Parameters
Background Color: Grid Color: ☑ Display Grid ☑ Display Block ID's ☐ Smooth Scrolling

Action Block Parameters
Width: 96 Height: 48 Color: Font: Arial - 10 point Connection:

Condition Block Parameters
Width: 96 Height: 64 Color: Font: Arial - 10 point True: False:

OptoScript Block
Width: 96 Height: 48 Color: Font: Arial - 10 point Connection:

Continue Block Parameters
Width: 96 Height: 48 Color: Font: Arial - 10 point

Text
Width: 192 Height: 128 Font: Arial - 10 point

Also Apply To
☐ ioControl
☐ Strategy
☐ All charts

2. Complete the fields as described in "Configure Chart Properties Dialog Box" below.

3. When you have made all the changes, click OK.

Configure Chart Properties Dialog Box

(A) Flowchart Properties Specify general chart display in this area. To apply these changes to existing charts as well as to new ones, click All charts in D.

- To change the chart background color from the default of white, click the Background Color box. Choose a new color from the Color dialog box.
- To change the chart's grid color from the default of black, click the Grid Color box.
- The grid and block ID numbers are displayed by default. To remove them, click Display Grid or Display Block ID's to remove the check mark.
- To enable or disable smooth scrolling in a flowchart, click Smooth Scrolling; this option is disabled by default.

(B) Action Block Parameters Define the appearance of action blocks, condition blocks, OptoScript blocks, and continue blocks in this area. These changes affect new blocks only, not existing blocks. (To change existing blocks, see "Changing Existing Elements to Match New Defaults" on page 8-5.)

- In the Width and Height fields, type the block size in pixels. For action and continue blocks, the default width is 96 and the default height is 48; the minimum parameters are 48 (width) and 32 (height). For condition blocks, the default height is 64. (Note that the numbers you enter are rounded down to be evenly divisible by 16; for example, if you enter 81, click OK and then reopen the dialog box, the parameter reads 80.)
- To change the color of the blocks from the default, click the Color box.
- To change block name text formatting, click the Font box. In the Font dialog box, change the font, font style, size, effects, and color. The default font is black 10-point Arial bold.
- To change text alignment, right-click the Font box and choose left, center, or right from the pop-up menu.
- To change the color of connection lines, click a Connection line at the far right. Choose the new color from the Color dialog box.

(C) Text Define width, height, and font of text blocks that appear as comments in a chart or subroutine. Default width is 192; default height is 128; the minimum for both is 16. The default font is black 10-point Arial bold.

(D) Also Apply To To expand the scope of the changes you’ve made, click these boxes. Click ioControl to apply the changes to all new strategies and subroutines in ioControl. Click Strategy to apply the changes to all new charts in the current strategy. Click All Charts to apply the changes to all new charts and all new graphic elements added to the current strategy.

Depending on which dialog box you are in and what is currently open, one or more of these options may be grayed out. For example, if you are in the Configure ioControl Default Properties dialog box, it is assumed that the changes are to be applied throughout ioControl, and that option is therefore grayed out.

(E) Reset All To reset all parameters and options to their factory default settings, click Reset All.

Changing Existing Elements to Match New Defaults

Once you have changed the defaults for the way elements appear in a chart, you can update existing blocks, connections, and text to match.

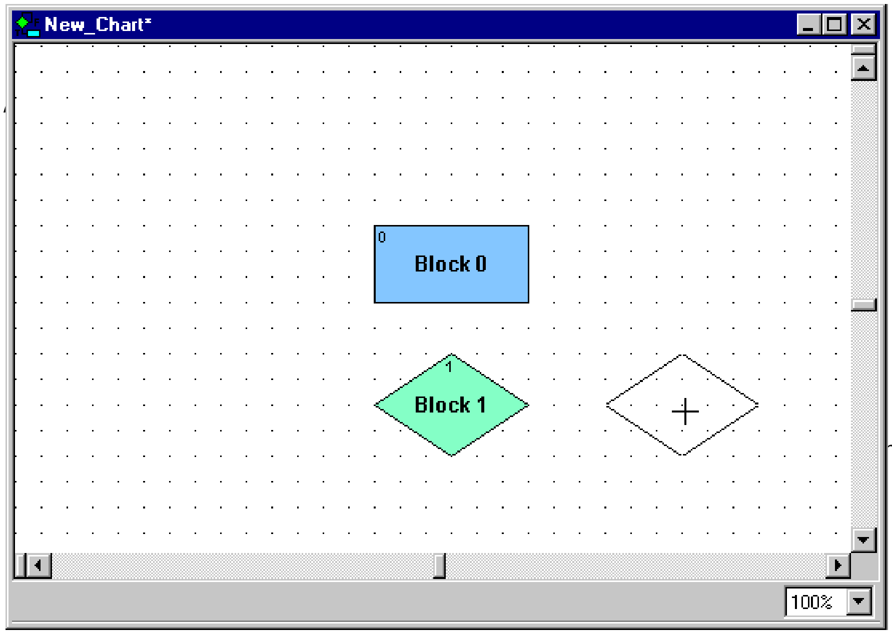
CAUTION: *When you update existing objects to match, you cannot undo the update.*

1. Right-click on an empty space in the chart whose elements you want to change. Choose Select from the pop-up menu. From the sub-menu, choose the item type you want.

   For example, to select all Action Blocks, choose Select→Action Blocks.

2. Right-click again in the chart, and choose Properties→Copy from Default from the pop-up menu.

   The color, size, and font of all selected items change to match the flowchart defaults.



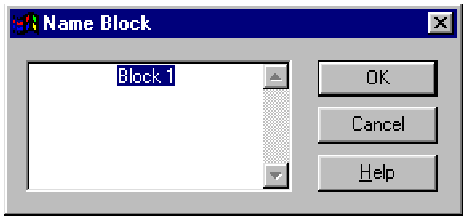
tool you want

he block.

3. Click in another location to place other blocks of the same type. When you have finished using the tool, click the right mouse button, click another tool in the toolbar, or press ESC.

Naming Blocks

1. With the chart open and the strategy in Configure or Online mode, click the Select tool and click the block to select it.
2. Right-click the block and choose Name from the pop-up menu.



Block 1

3. In the Name Block dialog box, type the name and click OK.

Renaming Blocks

1. With the chart open and the strategy in Configure or Online mode, click the Select tool and click the block to select it.
2. Right-click the block and choose Name from the pop-up menu.
3. In the Name Block dialog box, change the name. Click OK.

Connecting Blocks

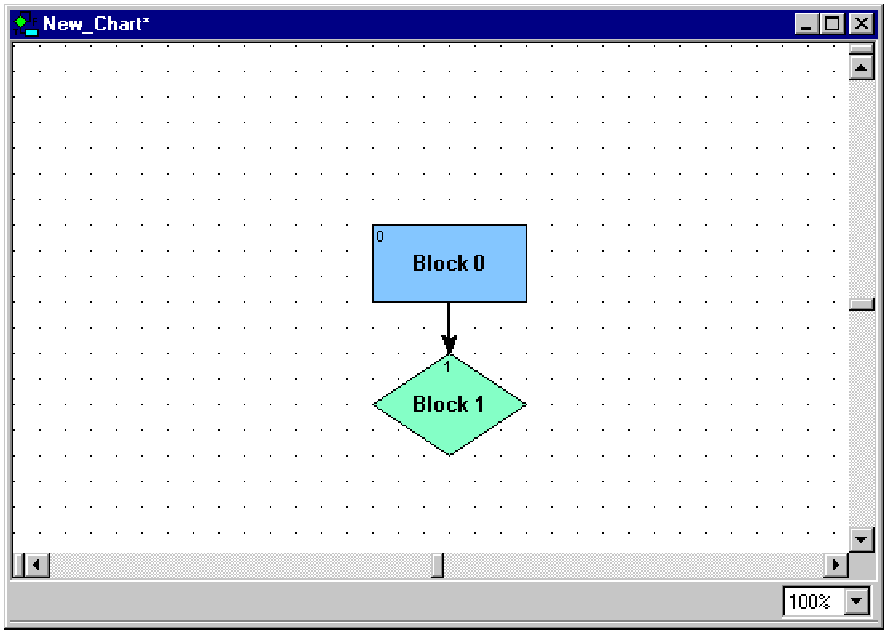
To connect blocks, start with the chart open and the strategy in Configure or Online mode. Remember that Action Blocks and OptoScript Blocks have only one exit, and Condition Blocks have two.

Action Blocks and OptoScript Blocks

1. To connect an action block or an OptoScript block to the next block in a program sequence, click the Connect tool ╲ .

2. First click the source block and then click the destination block.

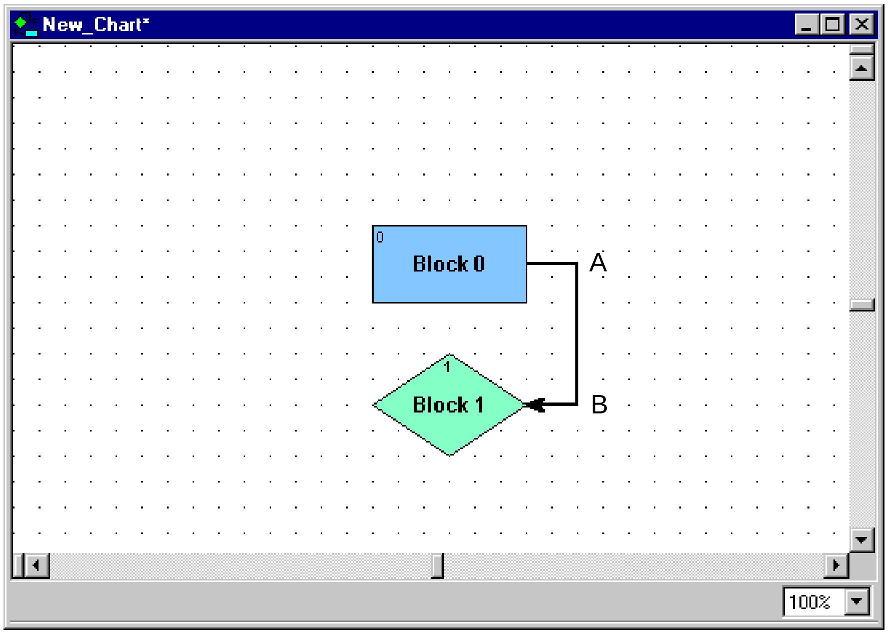
   Although you can click anywhere inside the blocks to make a connection, the connection is attached at the side closest to where you clicked. In the figure below, Block 0 is the source block and Block 1 is the destination block:



   To keep your charts neat, try to draw the most direct connections possible. To do so, after clicking the source block, move your cursor out of the block at a point closest to its destination.

3. To create a bend or elbow in a connection, click wherever you want the bend while drawing the connection.

For example, to draw the connection in the following figure, we selected the Connect tool, clicked Block 0, moved the cursor out of the block to the right, clicked at point A, clicked again at point B, and then clicked the right side of Block 1:

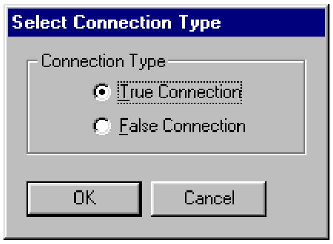


4. While you’re still drawing a line, to delete an elbow you don’t want, click the right mouse button once to undo it.

   If you created several elbows, you can eliminate them in reverse order with repeated right mouse clicks. If no more elbows remain and you right-click again, you delete the connection. Once you have completed a connection, however, you cannot undo it this way.

Condition Blocks

1. To connect a Condition Block to the next block in a program sequence, click the Connect tool ╲ .
2. Click the source block.



Select Connection Type

Connection Type

● True Connection

○ False Connection

3. Indicate whether you are drawing the True connection or the False connection, and then click OK.

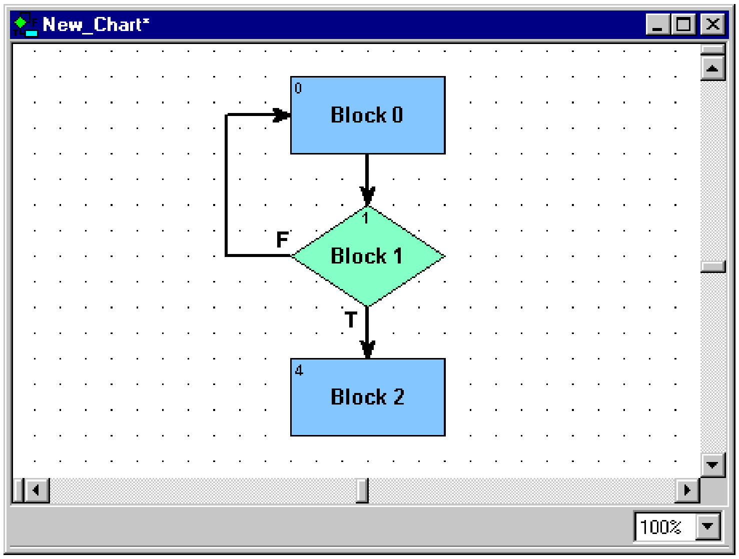
4. Click the destination block you chose (True or False).

   The connection is labeled T or F depending on its type.

5. Draw another connection to the second destination block.

   It is labeled the opposite exit type.

   For example, the following figure shows the True and False connections from the condition block, Block 1. If the conditions in Block 1 are true, Block 2 is executed next. If the conditions in Block 1 are false, Block 0 is executed next:



Adding Text

One of the best places to put comments about a Strategy is directly on its flowcharts. Start with the chart open and the strategy in Configure or Online mode.

1. To add text to a chart, click the Text tool A.

   When you move the mouse onto the chart, a rectangle representing the text area appears.

2. Click the mouse button and type your comments.

   If you type in more text than the text frame holds, it expands in length.

3. When you have finished typing, click anywhere on the chart outside the text frame.

   The frame becomes invisible, and only the text appears. To change the size or shape of the text block, see “Resizing Blocks or Text Blocks” on page 8-13.

4. Click in another location to create another text frame, or release the tool by right-clicking in the chart or choosing another tool from the toolbar.

Editing Text

1. With the chart open and the strategy in Configure or Online mode, click the Select tool .
2. Double-click the text block you want to change.

   A blinking cursor appears at the beginning of the text.
3. Change the text as needed.

   You can use any standard Windows CTRL key combinations when editing, including CTRL+arrow keys and CTRL+HOME or END for navigation. You can also use CTRL+X (cut), CTRL+C (copy), and CTRL+V (paste).
4. When you have finished changing text, click outside the text frame.

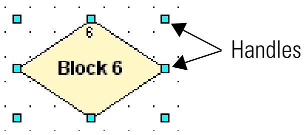
   The text block stays the same width but changes length to accommodate additional or deleted text. To change the size or shape of the text block, see "Resizing Blocks or Text Blocks" on page 8-13.

Selecting Elements

Before you can manipulate most elements, you need to select them. Start with the chart open and the Strategy in Configure or Online mode.

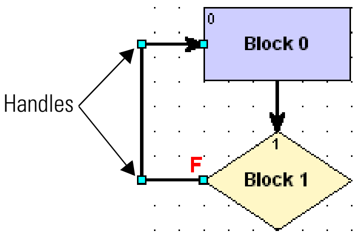
1. Click the Select tool .
2. To select an action, OptoScript, condition, continue, or text block, click the block.

   Handles appear around the block:



3. To select a connection, click it.

   Handles appear at the elbows and end points of the connection:



4. To select all connections entering or exiting a block, click the block, click the right mouse button, and choose Select Connections from the pop-up menu.

5. To select more than one element, do one of the following:
   - Select the first element, hold down the SHIFT key, and select additional elements.
   - Click and drag the mouse to draw a rectangle completely around the elements you want to select.
   - To select all items of the same type, right-click anywhere in the window and choose Select from the pop-up menu. From the sub-menu, choose the item type you want.

Moving Elements

1. With the chart open and the Strategy in Configure or Online mode, click the Select tool .

2. To move any action, OptoScript, condition, continue, or text block, click it. Then click and hold the mouse button anywhere on the selected item except on its handles, and drag it to the position you want.

   You can also use the arrow keys on your keyboard to move a block in any direction. Note that when you move a block, any connections attached to it also move.

3. To move a connection, click it. Then click and drag any handle in any direction.

   You can also move an end point from one block to another, as long as the result is a valid connection. A disallowed move is ignored.

4. To move several elements at once, select them, and then click and drag them.

   If elements end up stacked on top of each other, you may need to change their z-order before you can move them. See the following section.

Moving Elements in Front of or Behind Other Elements (Changing Z-Order)

If elements are stacked on top of each other, you can select only the one in front. To change their position (z-order), follow these steps:

1. Click the element to select it.

2. Right-click the element. From the pop-up menu, choose Z-order. From the sub-menu, choose the action you want to take:
   - Bring Forward—moves the element one position closer to the front.
   - Bring To Front—moves it all the way to the front.
   - Send Backward—moves the element one position closer to the back.
   - Send To Back—moves it all the way to the back.

Cutting, Copying, and Pasting Elements

You can cut, copy, and paste most chart or subroutine elements. Cut or copied elements are placed on the Windows Clipboard, and they can be pasted in the same chart or subroutine, in a different chart or subroutine, or in a different Strategy.

A connection can be cut or copied, but it cannot be pasted unless its original source and destination blocks have also been pasted. Block 0 cannot be cut.

1. With the chart open and the strategy in Configure or Online mode, click the Select tool ↖ .
2. To cut or copy element(s), click them. Press CTRL+X to cut or CTRL+C to copy.

   You can also select the element(s) and then choose Edit→Cut or Edit→Copy, or click the right mouse button and choose Cut or Copy from the pop-up menu.

3. To paste blocks, press CTRL+V, select Edit→Paste, or right-click anywhere on a chart and select Paste from the pop-up menu.

   Text blocks are pasted immediately. For action, condition, or continue blocks, a message appears asking if you want to keep the original name of the block being pasted.

   If you paste to a different strategy or to a subroutine, ioControl checks the referenced variables to make sure they match. Variables that do not exist are created. Variables that exist but are different—for example, a table with the same name but a different table length—are noted in a log file that appears when the paste is complete.

Deleting Elements

1. Make sure the chart is open and the strategy is in Configure or Online mode.
2. Click the Select tool ↖ . Click the element(s) to select them.

   CAUTION: *Make sure you have selected the element you want. You cannot undo a deletion!*

3. Press DELETE.

   You can also select the element(s), right-click them, and select Delete from the pop-up menu. Block 0 cannot be deleted.

Changing Element Color and Size

You can change the colors and sizes of blocks, connections, and text in your chart. To change one element (for example, the color of one block), use the steps in this section. To change more than one at a time, see "Changing the Appearance of Elements in a Chart Window" on page 8-3.

Start with the chart open and the strategy in Configure or Online mode.

Resizing Blocks or Text Blocks

1. Click the Select tool and click the block to select it.
2. Click one of the handles, then drag it in the direction you want. To resize horizontally and vertically at the same time, drag a corner handle.

Changing Block Colors

1. Click the Select tool and click the block to select it.
2. Right-click the block and choose Color from the pop-up menu.
3. Pick the color you want and click OK.

Changing Text

You can change the size, font, font style, or color of the text in any block.

1. Click the Select tool and click the block to select it.
2. Right-click the block and choose Font from the pop-up menu.
3. In the Font dialog box, make the changes you want. Click OK.
4. To change whether text appears at the left, the center, or the right of a block, select the block and click the right mouse button. From the pop-up menu, choose Justify; from the sub-menu, choose Left, Center, or Right.

Changing an Element Back to the Defaults

1. Select the item and click the right mouse button.
2. From the pop-up menu, choose Properties. From the sub-menu, choose Copy from Default.

   To change defaults, see "Changing the Appearance of Elements in a Chart Window" on page 8-3.

Opening, Saving, and Closing Charts

Opening a Chart

Make sure the Strategy is open. In the Strategy Tree, double-click the chart you want to open.

You can also open a chart by selecting Chart→Open, and then double-clicking the chart name in the Open Chart dialog box, which lists all charts that are not currently open.

If a chart is open but not visible on the screen, click the chart's name tab at the bottom of the window to make it visible.

Saving a Chart

1. Make sure the chart is open and is the active window.
2. From the Chart menu, choose Save.

Charts are automatically saved when you choose File→Save All. If you choose File→Save Strategy or click the Save Strategy button 💾 on the toolbar, you can choose which charts to save in the Save Strategy dialog box.

Closing a Chart

To close a chart, click the close box in the upper-right corner of the chart's window (not the ioControl window). You can also close a chart by pressing CTRL+F4 when the chart window is active. If you have made changes to the chart, you are prompted to save them.

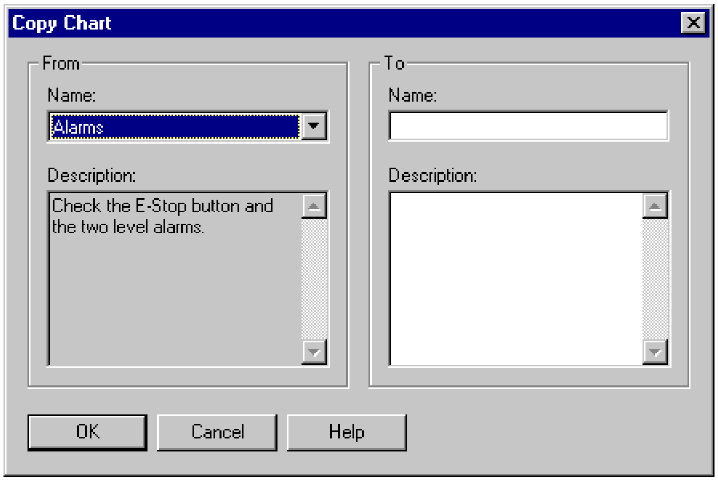
Copying, Renaming, and Deleting Charts

Copying a Chart

If an existing chart is similar to one you want to create, it is easier to copy it than to create a new one from scratch. To copy a chart in the same Strategy, follow the steps in this section. To copy a chart to another strategy, see "Exporting and Importing Charts" on page 8-17.

1. With the strategy open and in Configure mode, select Chart→Copy.

The Copy Chart dialog box appears.



2. In the From field, choose the chart you want to copy from the drop-down list.

3. In the To field, enter a name for the new chart.

   The name must start with a letter and include only letters, numbers, or underscores. (Spaces are converted to underscores).

4. (Optional) Enter a description for the new chart.

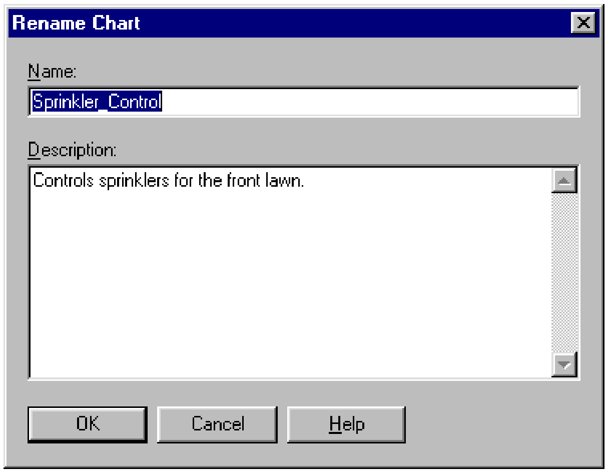
5. Click OK.

   The new chart is created and appears as the active window on the screen.

Renaming a Chart

1. Make sure the strategy is in Configure mode and that the chart you want to rename is the active window.

2. From the Chart menu, choose Rename.



Rename Chart

Name: Sprinkler_Control

Description: Controls sprinklers for the front lawn.

3. Enter a new name and description (optional). Click OK.

   The chart is renamed.

Deleting a Chart

You can delete any charts except for the Powerup chart. However, you cannot delete a chart if it is called or used by another chart in your Strategy.

1. Make sure the strategy is open and in Configure mode.

2. In the Strategy Tree, right-click the name of the chart you want to delete and choose Delete from the pop-up menu. Or, if the chart is the active window, choose Chart→Delete.

3. At the confirmation message, make sure you are deleting the correct chart.

   CAUTION: *You cannot undo a deletion!*

4. Click Yes to delete the chart.

   The chart window disappears (if it was open), the chart is removed from the Strategy Tree, and the strategy is saved.

Printing Charts

You can print any flowchart. To print a chart as it appears on the screen, see "Printing Chart or Subroutine Graphics" on page 7-23. To print commands (instructions) for the chart, see "Viewing and Printing Strategy or Subroutine Commands" on page 7-25.

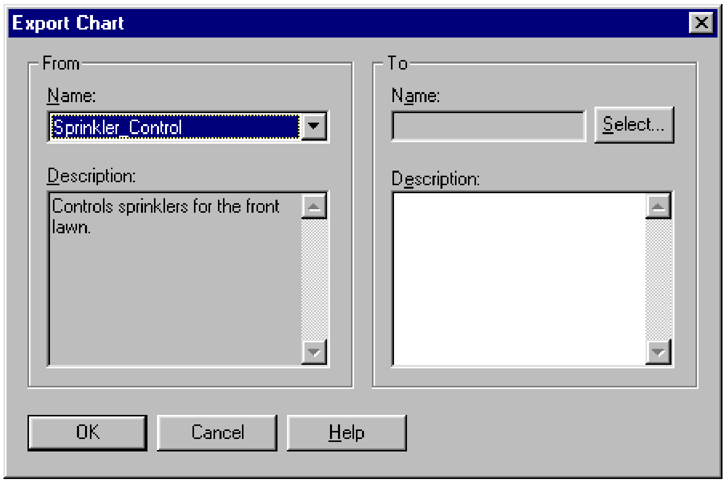
Exporting and Importing Charts

To copy a chart to another Strategy, you must export it as an ioControl chart export file (.cxf file) and then import it into the strategy where you want it.

Exporting a Chart

1. With the strategy open and in Configure or Online mode, choose Chart→Export.

   The Export Chart dialog box appears.

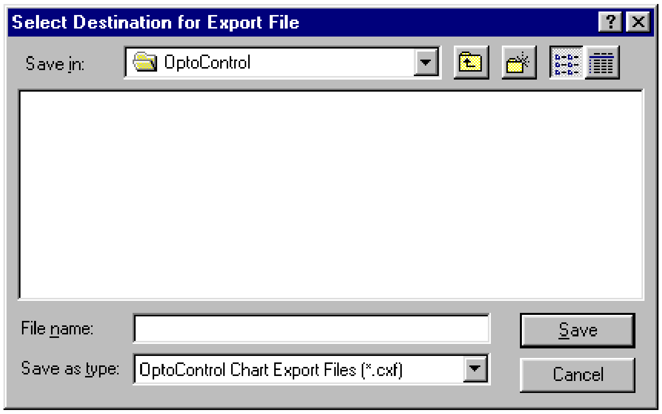


Name: Sprinkler_Control

Description: Controls sprinklers for the front lawn.

2. In the From section of the dialog box, select the chart to be exported from the Name drop-down list.

3. In the To section of the dialog box, click Select.



Save in: OptoControl

File name:

Save as type: OptoControl Chart Export Files (*.cxf)

4. Navigate to where you want the exported chart file to be saved. In the File name field, enter a name for the exported chart. Click Save.

   You return to the Export Chart dialog box, which now shows the path and file name in the To section.

5. (Optional) Enter a description for the new chart.

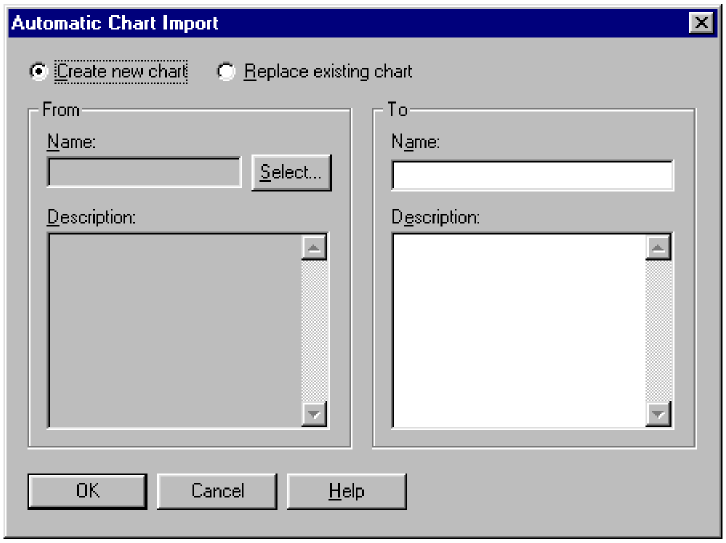
6. Click OK.

   The exported chart is saved. You can import it into any ioControl strategy. See the next section for information on importing charts.

Importing a Chart

1. With the strategy open and in Configure mode, choose Chart→Import.

   The Automatic Chart Import dialog box appears.



2. At the top of the dialog box, click Create new chart or Replace existing chart.

   CAUTION: *If you choose Replace existing chart, the old chart will be completely overwritten with the chart you are importing.*

3. Click Select. Navigate to the exported chart. Click OK.

4. In the To section of the dialog box, enter a name for the new chart. If you wish, enter a description. If you are replacing an existing chart, choose the chart to be replaced. Click OK.

   The chart is imported. A Chart Import Report window shows you how the tags in the chart match with those already in the strategy. Any tags from the chart that do not already exist in the strategy are created and added.

CHAPTER 9

Using Variables and Commands

Introduction

This chapter discusses the seven types of variables used in ioControl: numeric, string, pointer, numeric table, pointer table, string table, and communication handle variables.

This chapter also shows you how to use the commands, or instructions, in ioControl and discusses the mechanics of adding commands to your strategy flowcharts. For command information to help you program your strategy effectively, see Chapter 10. To find out how to use commands in OptoScript code, see Chapter 11. For a list of all standard ioControl commands and their OptoScript equivalents, see Appendix E.

In this Chapter

About Variables

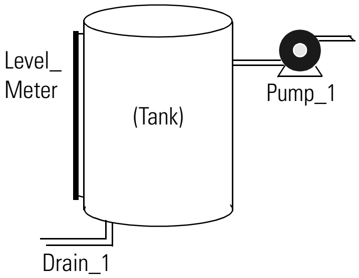
As Chapter 2 mentions, variables store pieces of information in a Strategy. You create a variable for each piece of information in your control process that must be acted upon. These pieces of information might include the name of a chart, the on or off state of a switch, communication parameters for a peer on the network, or a table that holds a series of numbers.

Each variable has a name and a value. You assign the variable's name in plain English, so you know what it is. The variable's value is the current information it represents. As a strategy runs, the variable's name remains the same, but its value may change. For example, the name of the variable Oven_Temperature stays the same, but its value (the temperature of the oven) may change several times while the strategy is running.

To illustrate variables, suppose you are regulating the amount of water in a tank. You must keep the tank filled beyond a minimum level, but you cannot let it get too full.

You've already configured the I/O points:

- Level_Meter is an analog Input Point that registers the quantity of water in the tank.
- Pump_1 is a digital Output Point that turns the pump on or off.
- Drain_1 is a digital output point that opens or closes the drain.



Next, you configure variables as places to hold information that these I/O points must work with:

- To avoid constantly polling Level_Meter to find out the quantity of water in the tank, you create a variable called Tank_Level_Reading in which to store the level. The input point Level_Meter is periodically checked and the value of Tank_Level_Reading updated.
- To establish the maximum and minimum levels, you create variables called Tank_Max_Level and Tank_Min_Level. The value in Tank_Level_Reading can be compared to the values in these two variables to determine whether the pump should be turned on or off, or the drain opened or closed, to maintain the proper level. (You could create constant values, called *literals*, for the minimum and maximum levels, but creating them as variables lets you change their values in Debug mode.)

Types of Data in a Variable

A variable stores one of six types of data: floating point, integer, timer, string, pointer, or communication handle. When you create the variable, you designate the type of data it contains. It is always best to choose the most appropriate data type for the information you are storing. ioControl can store an integer in a floating point variable, but it has to convert the data first. Unnecessary data conversions take up processor time.

- **Numeric** data stores numbers and can be one of the following types:
  - A **floating point** (or *float*) is a numeric value that contains a decimal point, such as 3.14159, 1.0, or 1,234.2. A good example of a float variable is one that stores readings from an analog input such as a thermocouple. ioControl uses IEEE single-precision floats with rounding errors of no more than one part per million.
  - An **integer** is a whole number with no fractional part. Examples of integer values are -1, 0, 1, 999, or -4,568. The state of a switch, for example, could be stored in an integer variable as 1 (on) or 0 (off).

    Most integers used in ioControl are 32-bit signed integers, which can range from -2,147,483,648 to 2,147,483,647. These 32-bit integers should be used for general integer use, such as status variables, mathematics, and indexes. If you are using the

SNAP-UP1-D64, SNAP-UP1-M64, or SNAP-ENET-D64 brains, which handle digital modules in all 16 module positions, you can use 64-bit integers to address the entire I/O unit at once. Integer 64 commands are slower than integer 32 commands and should be used only for these brains.

  - A **timer** stores elapsed time in units of seconds with resolution in milliseconds. Up Timers count up from zero, and Down Timers start from a value you set and count down to zero. Timers can range from 0.001 to 4.611686 x 10^{15}.

- A **string** stores text and any combination of ASCII characters, including control codes and extended characters. For instance, a string variable might be used to send information to a display for an operator to see. A string variable is also used to set parameters for peer-to-peer communication. When defining most string variables, you must specify the width of the string. The width is the maximum number of characters that the variable may hold.

  A string variable can contain numeric characters, but they no longer act as numbers. To use them in calculations, you must convert them into floating point or integer numbers. Conversely, a numeric value to be displayed on a screen must first be converted into a string.

- A **pointer** does not store the value of a variable; instead, it stores the memory address of a variable or some other ioControl item, such as a chart or an I/O point. You can perform any operation on the pointer that you could perform on the object the pointer points to. Pointers are an advanced programming feature and are very powerful, but they also complicate programming and debugging.

- A **communication handle** stores information used to communicate with other entities, for example other devices on the network or files that store data. The communication handle's value is a string containing the parameters needed to make a connection with a specific entity. For outgoing Ethernet communication, for example, these parameters usually include the protocol, the IP address of the device you are communicating with, and the port number used on the device.

  After the initial connection is made, the communication handle is referenced during communication with the entity, and then used to close communication.

Variables in ioControl

In ioControl there are seven types of variables:

- numeric
- numeric table
- string
- string table
- pointer
- pointer table
- communication handle

Numeric, string, and pointer variables contain individual pieces of data. Numeric table, string table, and pointer table variables contain several pieces of related data in the form of a table. Communication handle variables contain parameters used by ioControl for communicating with other devices and files.

Table Variables

In a table variable, the variable name represents a group of values, not just one. Table variables are one-dimensional arrays, which means that each table is like a numbered list of values. You refer to each item in the list by its number, or *index*. Indexes start at 0, not at 1. Here are two table examples:

Index	Value
0	82.0
1	86.1
2	85.0
3	74.8
4	72.3
5	72.7

In a *float table,* values are floating point numbers.

Index	Value
0	Maria
1	Tom
2	Siu
3	Andre

In a *string table,* values are strings.

When you define a table, you must specify its length, which is how many values the table can store. The length of a table is NOT the value of the last index. Since indexes start at 0, a table with a length of 100 contains indexes 0 through 99. Table length is limited only by the amount of memory in the control engine. (For more information on the available memory, see "If You Have Memory Problems" on page A-3.)

Numeric tables store either integer values or floating point numbers (but not both in the same table). String tables store strings. Because pointers can point to any data type (for example, an I/O point, a chart, or even another table), pointer tables can store an assortment of data types.

Persistent Data

Most variables can be either *global* or *persistent* in scope. Global variables are set to an initial value (which you specify during configuration) either whenever the Strategy is run or whenever it is downloaded.

Persistent variables, however, are initialized only when the strategy is first downloaded. The variable's value is saved in the controller's memory; it does not change when the strategy is run, stopped, or started, and it does not change if the strategy is changed and downloaded again. A persistent variable's value remains the same until one of the following events occurs:

- A strategy with a different name is downloaded.
- The RAM memory on the controller is cleared.
- A new firmware kernel is downloaded to the controller.

- The persistent object is changed in some way (for example, the length of a persistent table changes).

Persistent data can be very useful in certain situations. For example, suppose you have a PID setpoint that is fine-tuned as your process runs. If the setpoint is configured as persistent, its value won't be lost if you must re-download the strategy after making a program change.

Pointer variables, pointer tables, and timers are not allowed to be persistent; but all other variables and tables can be. Persistent variables cannot be configured on-the-fly.

Literals

A literal is used like a variable, but it is constant data that never changes. A literal has no variable name, only a fixed value, which may be a floating point number, an integer, or a string.

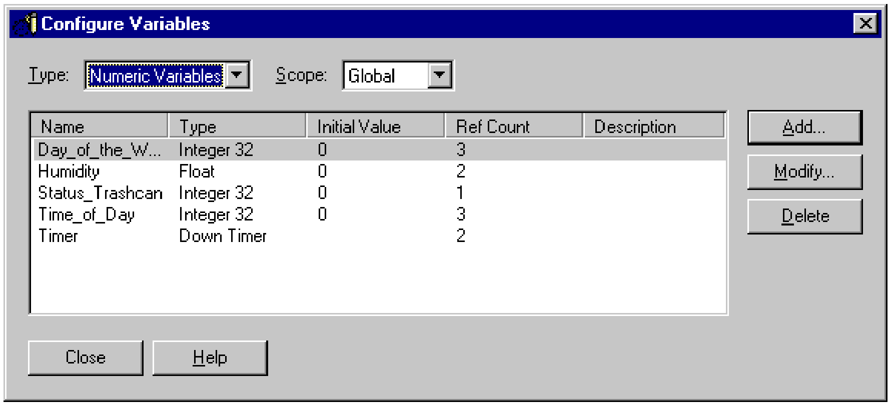
If you are using subroutines, see the information on literals in "Data Types for Subroutines" on page 12-2.

Adding Variables

This section includes steps for adding numeric, string, pointer, and communication handle variables. (For numeric, string, and pointer tables, see "Adding Tables" on page 9-8.)

1. With the strategy or subroutine open in Configure mode, click the Configure Variables button on the toolbar or choose Configure→Variables.

The Configure Variables dialog box opens.



Configure Variables

Type: Numeric Variables

Scope: Global

Name	Type	Initial Value	Ref Count	Description
Day_of_the_W...	Integer 32	0	3	
Humidity	Float	0	2	
Status_Trashcan	Integer 32	0	1	
Time_of_Day	Integer 32	0	3	
Timer	Down Timer		2	

This dialog box lists all the variables in the strategy or subroutine that are of the type shown in the Type field.

2. In the Type drop-down list, choose the type of variable you want to configure.

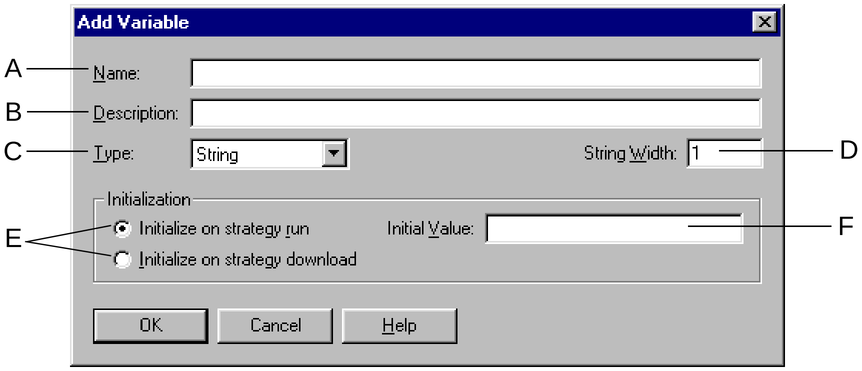
3. To have data in the variable be persistent, select Persistent in the Scope drop-down list.

You must choose Persistent as the scope *before* creating the variable; existing variables cannot be changed to be persistent. Pointer variables and timers cannot be persistent. Global is the default scope. For more information, see "Persistent Data" on page 9-4.

4. If you are adding the variable to a subroutine, select Subroutine in the Scope drop-down list.

5. To add a new variable, click Add.

   The Add Variable dialog box appears.



Field at **D** varies depending on variable type.

The figure above shows the Add Variable dialog box as it appears for string variables. Fields are slightly different for other variables.

6. Complete the fields as described in "Add Variable Dialog Box" below.

7. Click OK.

   The Add Variable dialog box closes and the new variable appears in the Configure Variables dialog box.

Add Variable Dialog Box

This dialog box varies depending on the type of variable.

(A) Name Enter a name for the variable. The name must start with a letter and may contain letters, numbers, and underscores. (Spaces are converted to underscores.)

(B) Description (Optional) Enter a description of the variable.

(C) Type In the Type drop-down list, select the type of data for the variable. Possible types are shown in the following table. For more information, see "Types of Data in a Variable" on page 9-2.

Variable	Possible Data Types
Numeric	Integer 32, integer 64, floating point, up or down timer
Numeric table	Integer 32, integer 64, or floating point

Variable	Possible Data Types
String	String
String table	String
Pointer	Pointer to any data type
Pointer table	Pointer to any data type, and the data type may change over time
Communication handle	Communication handle

(D) String Width

This field varies depending on the variable type.

(**Table variables** only) In the Table Length field, enter an integer between 1 and 1,000,000 representing the number of elements in the table. The greater the number, the more memory required to store the table.

(**String variables** and **string tables** only) In the String Width field, enter the maximum number of characters permitted in the string. The number must be an integer between one and 1024. The greater the number, the more memory required to store the string.

(**Pointer variables** only) From the Pointer to Type drop-down list, select the type the pointer points to. Note that void pointers are not allowed; a pointer must point to a specific type. Also note that you cannot point a pointer to another pointer; ioControl has only one level of indirection for pointers. If you try to point to a pointer, ioControl assigns to the new pointer the address of the object being pointed to.

(E) Initialization (For all variables except up timers and down timers) To set the variable to the initial value (F) each time the strategy is run (either manually from the debugger or automatically via the autorun flag), click Initialize on Strategy Run.

To set the variable to the initial value (F) only when a strategy is downloaded, click Initialize on Strategy Download. The variable retains its current value when the strategy is stopped and then run again, either through the debugger or autorun. It also retains its value if power is cycled. Note that this choice means the variable is stored in battery-backed RAM, which is limited in size. See "If You Have Memory Problems" on page A-3. To keep a variable's current value through both power cycles and strategy download, make the variable a persistent variable (see "Persistent Data" on page 9-4).

The following table shows how your choices about variable initialization and persistence affect what happens to variables (applies to all variables and tables except up timers and down timers).

Variable	What happens to the variable's value when...		
	Strategy stops, then starts (through debugger or autorun)	Power is cycled	Same strategy is downloaded
Variable initialized on strategy run (default)	Set to initial value	Set to initial value	Set to initial value
Variable initialized on strategy download	Retains current value	Retains current value	Set to initial value
Persistent variable	Retains current value	Retains current value	Retains current value

(F) Initial Value (For all variables except pointers, up timers, and down timers) Enter the value to which the variable is to be set initially. If you leave this field blank, the initial value is set to zero.

(For **pointer variables** only) When you have selected the Pointer to Type, the drop-down list in this field shows all the valid objects for that type that are currently defined in your strategy. Choose one or leave the initial value as NULL. A NULL value means that the pointer is created but does not initially point to anything.

(For **communication handles** only) Enter a string containing communication parameters in the correct format for the type of communication handle you are using. The type (for example, `tcp`, `ftp`, `file`) must be in lowercase letters, and parameters are separated by colons and commas according to the format required. See "Communication Commands" on page 10-35 for information and examples.

- If you are talking to a serial module, use the IP address of the brain the module is attached to, and use the serial module's port number according to its position on the rack, for example: tcp:10.192.55.185:22502 See Opto 22 form #1191, the *SNAP Serial Communication Module User's Guide*, for port numbers.
- If you are talking to another SNAP Ultimate brain, be aware of port numbers that are reserved for a specific protocol. For more infomration, see the section on Security in form #1311, the *SNAP Ultimate I/O System User's Guide*.
- For other peers on the Ethernet network, be aware of port numbers they may use for specific purposes. Ports 22000 and 22001 are reserved for the control engine. For a list of standard Ethernet port numbers, refer to http://www.iana.org/assignments/port-numbers.
- If you are using the Accept Incoming Communication command to listen for communication requests, leave out the IP address and use the following format: tcp:port

For tables, each value in the table is set to the initial value. If you need to set individual table elements to differing values intially, you can do so in the Powerup chart. If you need to use an initialization file to set values on strategy download, contact Opto 22 Product Support.

Adding Tables

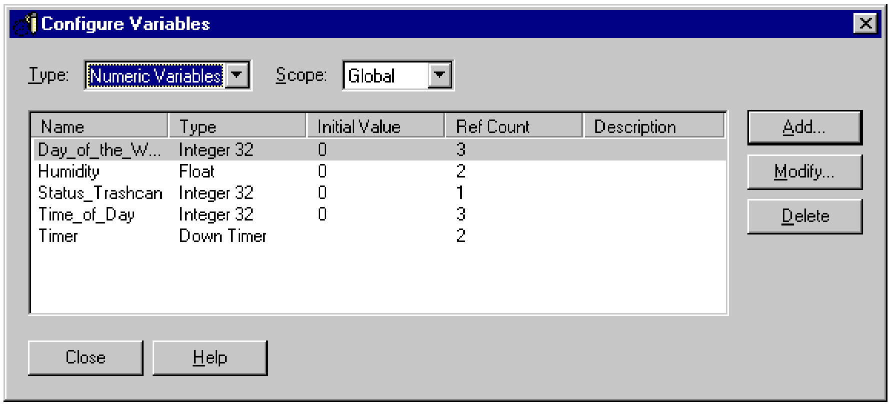
See the following topics to add numeric, string, and pointer tables:

- "Adding Table Variables" (below)
- "Setting Initial Values in Tables During Strategy Download" on page 9-10

Adding Table Variables

1. With the strategy or subroutine open in Configure mode, click the Configure Variables button 123 ABC on the toolbar or choose Configure→Variables.

   The Configure Variables dialog box opens, listing all variables of one type in the strategy.



Type: Numeric Variables

Scope: Global

Name	Type	Initial Value	Ref Count	Description
Day_of_the_W...	Integer 32	0	3	
Humidity	Float	0	2	
Status_Trashcan	Integer 32	0	1	
Time_of_Day	Integer 32	0	3	
Timer	Down Timer		2	

2. In the Type drop-down list, choose the type of table variable you want to add.

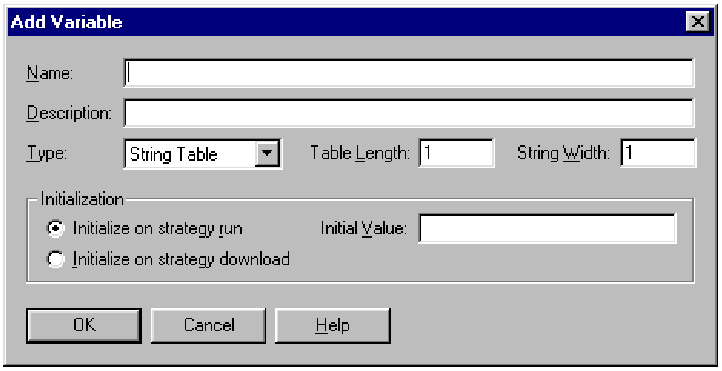
3. To have data in the table be persistent, select Persistent in the Scope drop-down list.

   Pointer tables cannot be persistent. Global is the default scope. For more information, see "Persistent Data" on page 9-4.

4. If you are adding the table variable to a subroutine, select Subroutine in the Scope drop-down list.

5. To add a new table variable, click Add.

The Add Variable dialog box appears.


Add Variable

Name:

Description:

Type: String Table

Table Length: 1

String Width: 1

Initialization

● Initialize on strategy run

○ Initialize on strategy download

Initial Value:

The figure above shows the dialog box as it appears for string tables. Fields are slightly different for other table variables.

6. Complete the fields as described in "Add Variable Dialog Box" on page 9-6.

   NOTE: If you need to set individual table elements to differing values initially, you can do so in the Powerup chart. If you need to use an initialization file to initialize the values on strategy download, see the next section, "Setting Initial Values in Tables During Strategy Download."

7. Click OK.

   The Add Variable dialog box closes and the new table variable appears in the Configure Variables dialog box.

Setting Initial Values in Tables During Strategy Download

When you are adding table variables in ioControl, you can set all table elements to one initial value in the Add Variables dialog box. If you want to set each individual table element to its own value, however, you need to create an initialization file and download it with your ioControl Strategy.

In addition to setting initial values for table elements, sometimes it is easier to initialize all variables during strategy download using an initialization file.

This section shows you how to create an initialization file and download it with your strategy.

Creating the Initialization File

A sample initialization file, INIT.TXT, is included with ioControl. You can open this file with a text editor to see the proper syntax for each initialization entry, and then modify it as necessary for your strategy.

IMPORTANT: *Every initialization file should include the following line near the top:*

```
""DOWNLOAD_COMPRESSION_OFF
```

Download compression removes the newlines necessary for string initialization. This line tells ioControl to turn off download compression so strings can be initialized.

IMPORTANT: *Each file must end with a carriage return, and each line within the file must end with a carriage return. If you add comments to the file, they must be preceded by a backslash (\) and a space.*

Text Examples

NOTE: Variable names are case-sensitive and can include both upper- and lower-case letters.

Variables Example To set initial values of 123 for the variable INTEGER_VARIABLE, 456.789 for FLOAT_VARIABLE, "String Variable Test String" for STRING_VARIABLE, and to have pointer PTR_POINTER_VARIABLE point initially to INTEGER_VARIABLE, you would include the following text in the initialization file:

```
123 ^INTEGER_VARIABLE @!
456.789 ^FLOAT_VARIABLE @!
*STRING_VARIABLE $INN
String Variable Test String
^INTEGER_VARIABLE MoveToPointer PTR_POINTER_VARIABLE
```

Integer Table Example To set initial values of 10, 20, 30, 40, and 50 for elements zero through four of an integer table named My_Int_Table, include the following text:

```
10 0 }My_Int_Table TABLE!
20 1 }My_Int_Table TABLE!
30 2 }My_Int_Table TABLE!
40 3 }My_Int_Table TABLE!
50 4 }My_Int_Table TABLE!
```

Float Table Example For a float table, the initial values must include a decimal point. To set initial values of 1.1, 2.2, 3.3, 4.4, and 5.5 for elements zero through four of a float table named My_Float_Table, include the following text:

```
1.1 0 }My_Float_Table TABLE!
2.2 1 }My_Float_Table TABLE!
3.3 2 }My_Float_Table TABLE!
4.4 3 }My_Float_Table TABLE!
5.5 4 }My_Float_Table TABLE!
```

String Table Example To set initial values of "zero", "one", "two", "three", and "four" for elements 0–4 of a string table named My_String_Table, include the following text. Make sure you turn off download compression and use new lines as shown:

```
0 {My_String_Table $TABLE@ $INN
zero
1 {My_String_Table $TABLE@ $INN
one
2 {My_String_Table $TABLE@ $INN
two
```

```
3 {My_String_Table $TABLE@ $INN
three
4 {My_String_Table $TABLE@ $INN
four
```

Pointer Table Example Each index in a pointer table points to another item within the strategy, for example an I/O point, a variable, or a chart. Setting initial values for pointer tables means designating the items the pointer table initially points to. For example, you would include the following text to have a pointer table named My_Ptr_Table initially point to Oven_Temperature (a variable), Alarm_Handler (a chart), Thermocouple (an analog input), Fuel_Pump (an analog output), and Fan_1 (a digital output):

```
^Oven_Temperature 0 PTBL_My_Ptr_Table TABLE!
&Alarm_Handler 1 PTBL_My_Ptr_Table TABLE!
~Thermocouple 2 PTBL_My_Ptr_Table TABLE!
~Fuel_Pump 3 PTBL_My_Ptr_Table TABLE!
~Fan_1 4 PTBL_My_Ptr_Table TABLE!
```

Special Characters in the Initialization File Note that the initial character on each line of the initialization file is a special character that identifies the object type. Possible characters include the following:

Character	Object Type
^	float, integer, or timer
*	string
PTR_	pointer variable
~	I/O point
&	chart

Character	Object Type
}	float table or integer table
{	string table
PTBL_	pointer table
%	I/O unit

Saving the Initialization File When you have finished modifying the file, save it as a text file (.txt file extension) to your strategy directory. You can name the file anything you like. (You can also save it to any directory you like, but it is good practice to save each initialization file to the directory of the strategy that references it.)

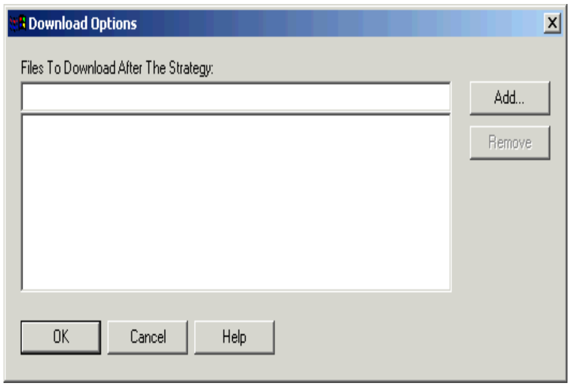
Downloading the Initialization File

To use the file, you need to download it with your Strategy.

1. With the strategy open in Configure or Online mode, click the Configure Control Engines button on the toolbar, choose Configure→Control Engines, or double-click the control engine's name.

2. Highlight the control engine and click the Download Options button.

   The Download Options dialog box appears. The initialization file must be downloaded immediately after your strategy is downloaded.



3. Click Add.

4. Navigate to the initialization file you created. Double-click the file name.

   The file appears in the Download Options dialog box.

5. Click OK.

   The initialization file sets values for the variables and table elements immediately after your next full strategy download.

   IMPORTANT: *When you create each variable in the strategy, you must check "Initialize on Strategy Download" in the Add Variable dialog box.*

   Since download options are specific to each control engine, make sure you set the initialization file as a download option for every control engine on which the strategy will run. Because control engines have separate initialization files, you can use the same master strategy for two or more control engines and configure differences in variables by customizing the initialization files.

Changing a Configured Variable

You can change a variable, but you cannot change a variable's type.

1. To change a configured variable, make sure the strategy or subroutine is open and in Configure or Online mode.

2. If you are working in a strategy, on the Strategy Tree, expand the Variables folder until you see the name of the variable you want to change. Double-click its name to open the Edit Variable dialog box.

3. If you are working in a subroutine, click the Configure Variables button 123 ABC on the toolbar or choose Configure→Variables. In the Configure Variables dialog box, double-click the name of the variable you want to change.

4. In the Edit Variable dialog box, make the necessary changes and click OK.

If you need help, see "Adding Variables" on page 9-5 or "Adding Tables" on page 9-8.

You can also change a variable from the Configure Variables dialog box by double-clicking the variable's name or by highlighting it and clicking Modify.

Deleting a Variable

You cannot delete a variable that is referenced within the Strategy or subroutine. *Be careful when deleting variables, since you cannot undo a deletion.*

1. Make sure the strategy is open and in Configure mode.
2. On the Strategy Tree, expand the Variables folder until you see the name of the variable you want to change. Right-click the name and choose Delete from the pop-up menu.

   The variable is deleted.

You can also delete a variable from the Configure Variables dialog box by highlighting the variable's name and clicking Delete.

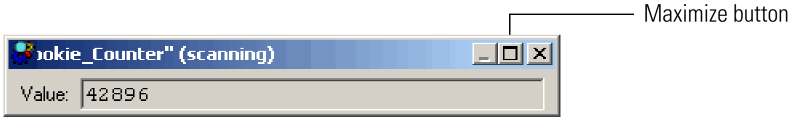
Viewing Variables in Debug Mode

While the Strategy is running in Debug mode, you can view its variables and modify the value of a variable or of entries in a table. You can also view several variables at once—as well as other strategy elements—by putting them into a watch window.

Viewing Numeric, String, and Communication Handle Variables

1. Make sure the strategy is running in Debug mode. On the Strategy Tree, double-click the variable you want to view.

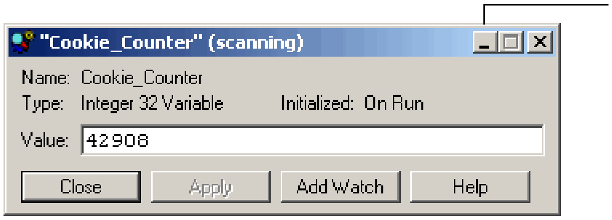
   The Inspect Variables dialog box opens. The animated icon at the upper left assures you that the data is fresh. The title bar includes the name of the variable and indicates whether scanning is occurring.



ɔokie_Counter" (scanning)

Value: 42896

2. To view more information, click the Maximize button.



"Cookie_Counter" (scanning)

Name: Cookie_Counter

Type: Integer 32 Variable Initialized: On Run

Value: 42908

Minimize button

Communication handle variables show slightly different fields.

Scanning stops whenever you click a changeable field. It resumes once you click Apply, another button, or an unchangeable field. If scanning resumes before you click Apply, any changes you made are lost.

If you do not want to change the value of the variable, you can click the Minimize button to shrink the dialog box back to its original size.

3. To change the value of the variable, type the new value in the Value field and click Apply.

The field turns magenta until you click Apply.

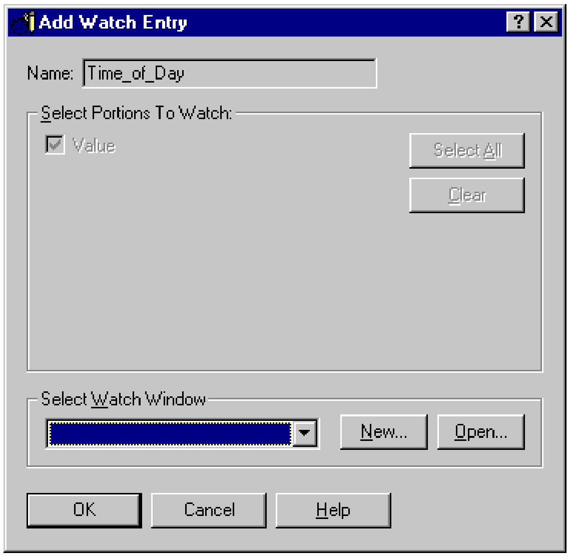
For a string variable, if your change lengthens the string beyond its maximum width, the string is truncated to fit.

For a communication handle variable, changing the value of the variable here has the same effect as using a Set Communication Handle Variable command. If the communication handle is currently open, the value will be changed *but will not affect the connection.*

4. To monitor the variable in a watch window, click Add Watch.

If you have only one watch window and it is already open, the variable appears immediately in the window for monitoring.

Otherwise, the Add Watch Entry dialog box appears.



Add Watch Entry

Name: Time_of_Day

Select Portions To Watch:

☑ Value

5. Check the items you want to watch.

Items to watch vary depending on the variable type.

6. In the Add Watch Entry dialog box, do one of the following:
   - If the watch window you want to use to monitor the variable is open, choose it from the Select Watch Window drop-down list.
   - If the watch window you want is not open, click Open. Navigate to it and double-click it to open it.
   - If you want to monitor the variable in a new watch window, click New. (For help, see "Creating a Watch Window" on page 6-73.)

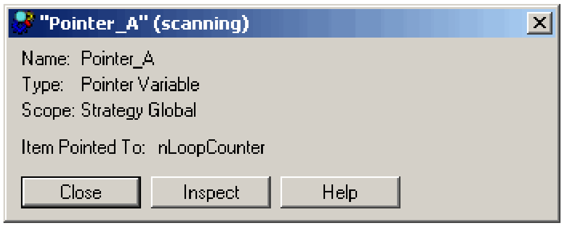
7. When the Add Watch Entry dialog box shows the correct items to watch and the watch window you want, click OK.

   The variable appears in the watch window.

Viewing Pointer Variables

1. Make sure the strategy is running in Debug mode. On the Strategy Tree, double-click the pointer variable you want to view.

   The View Pointer dialog box appears, showing the pointer's name, type, scope, and item pointed to.



"Pointer_A" (scanning)

Name: Pointer_A
Type: Pointer Variable
Scope: Strategy Global

Item Pointed To: nLoopCounter

2. To view the status or value of the item pointed to, click the Inspect button.

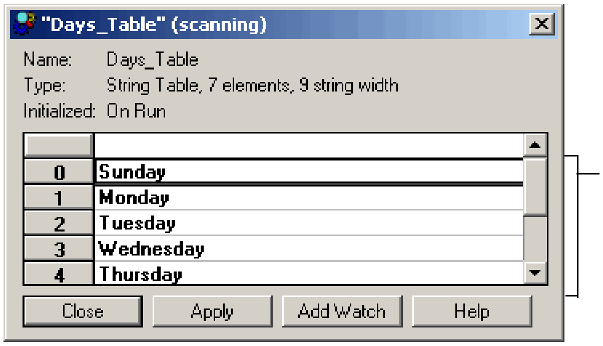
   If you need help, follow the steps in "Viewing Numeric, String, and Communication Handle Variables" on page 9-14.

Viewing Numeric and String Tables

1. Make sure the strategy is running in Debug mode. On the Strategy Tree, double-click the table variable you want to view.

   The View Table dialog box appears, showing the table's name, length (maximum number of entries), width for a string table, initialization method, and security level. It also lists the index and value of each table entry. The title bar includes the name of the variable and indicates whether scanning is occurring.

Here's an example for a string table.



Name: Days_Table
Type: String Table, 7 elements, 9 string width
Initialized: On Run

0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday

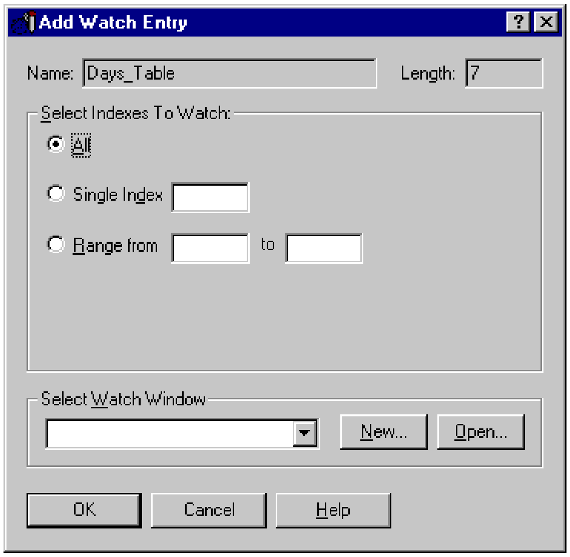
Table entries and their current values.
Resize the dialog box to see all entries at once.

Scanning for an individual table element stops whenever you select an element in the table. It resumes for that element if no changes are made and another table element is selected, or when you click Apply. A magenta background indicates that scanning is stopped.

2. To change a table entry, click its index number, highlight the value, and type in a new value. Click Apply.

3. To monitor the table in a watch window, click Add Watch.

   The Add Watch Entry dialog box appears.



Name: Days_Table Length: 7

● All
○ Single Index
○ Range from to

4. In the Add Watch Entry dialog box, do one of the following:
   - If the watch window you want to use to monitor the table variable is open, choose it from the Select Watch window drop-down list.
   - If the watch window you want is not open, click Open. Navigate to it and double-click it to open it.
   - If you want to monitor the variable in a new watch window, click New. (For help, see "Creating a Watch Window" on page 6-73.)

5. Select the indexes you want to watch.

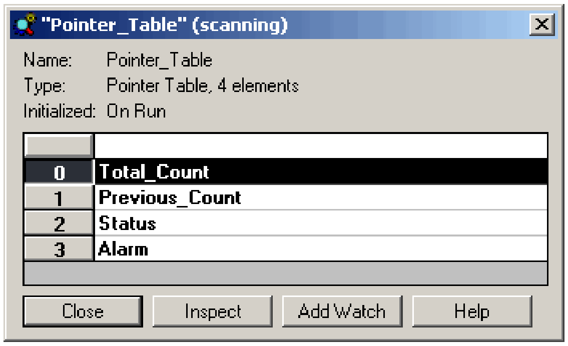
6. When the Add Watch Entry dialog box shows the correct items to watch and the watch window you want, click OK.

   The table variable appears in the watch window.

Viewing Pointer Tables

1. Make sure the strategy is running in Debug mode. On the Strategy Tree, double-click the pointer table you want to view.

   The View Table dialog box appears, showing the pointer table's name, length, and the items pointed to. You cannot change a pointer table entry in this dialog box.



Name: Pointer_Table
Type: Pointer Table, 4 elements
Initialized: On Run

0	Total_Count
1	Previous_Count
2	Status
3	Alarm

2. To view the status or value of the item pointed to, highlight it in the table and click the Inspect button.

   If you need help, follow the steps in "Viewing Numeric and String Tables" on page 9-16.

Adding Commands

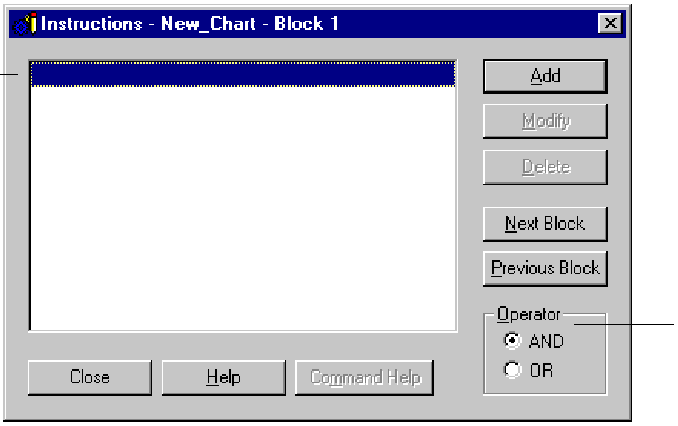
To make a block in a Strategy flowchart do the work it's intended to do, you add one or more commands. Commands use the I/O points and variables you've already configured, as well as other elements in your strategy. A command, for example, might turn on a Digital Point, move a value to a variable, or check to see whether a chart is running. ioControl contains more than 500 commands you can use. A command in ioControl is often called an instruction.

You can add commands to Action Blocks, Condition Blocks, and OptoScript Blocks. Continue blocks just move flowchart logic to another block. (See page 9-25 for steps to configure a continue block.)

To add commands to an action or condition block, follow the steps below. (To add commands to an OptoScript block, see "OptoScript Functions and Commands" on page 11-11 and "Using the OptoScript Editor" on page 11-25.)

1. With the strategy open in Configure or Online mode and the flowchart open, double-click the block to which you want to add a command.

   The Instructions dialog box appears. The highlighted area shows where the new command will be added.

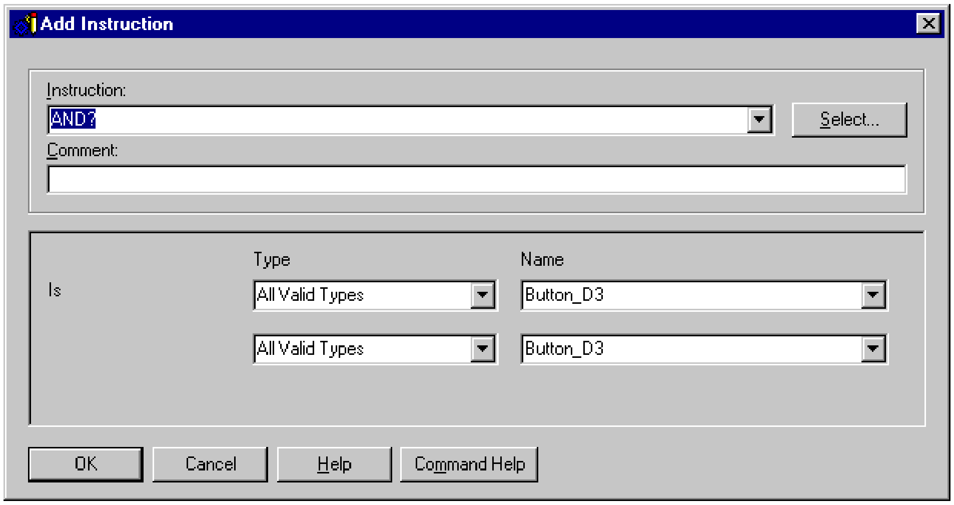


Operator: ● AND ○ OR

Highlighted area

The Operator group appears in the dialog box only for condition blocks, not for action blocks.

2. Click Add to open the Add Instruction dialog box.



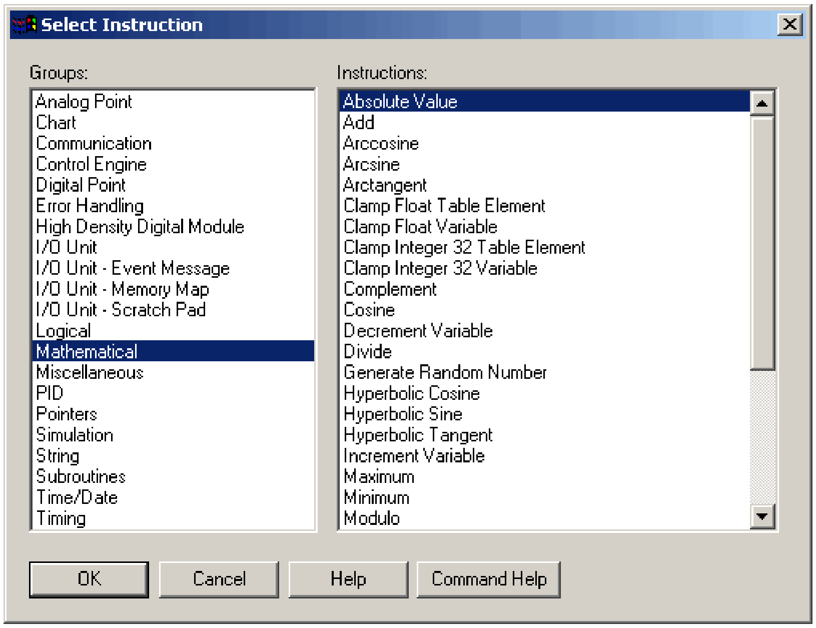
Instruction: AND?

Comment:

Is	Type	Name
	All Valid Types	Button_D3
	All Valid Types	Button_D3

3. If you know the command you want, enter its name in the Instruction field by typing it in or by choosing it from the drop-down list. Skip to step 7.

4. If you don't know the command name, click Select to open the following dialog box.



5. Click the name of a command group in the left column to display all the commands in that group. In the right column, click the command you want.

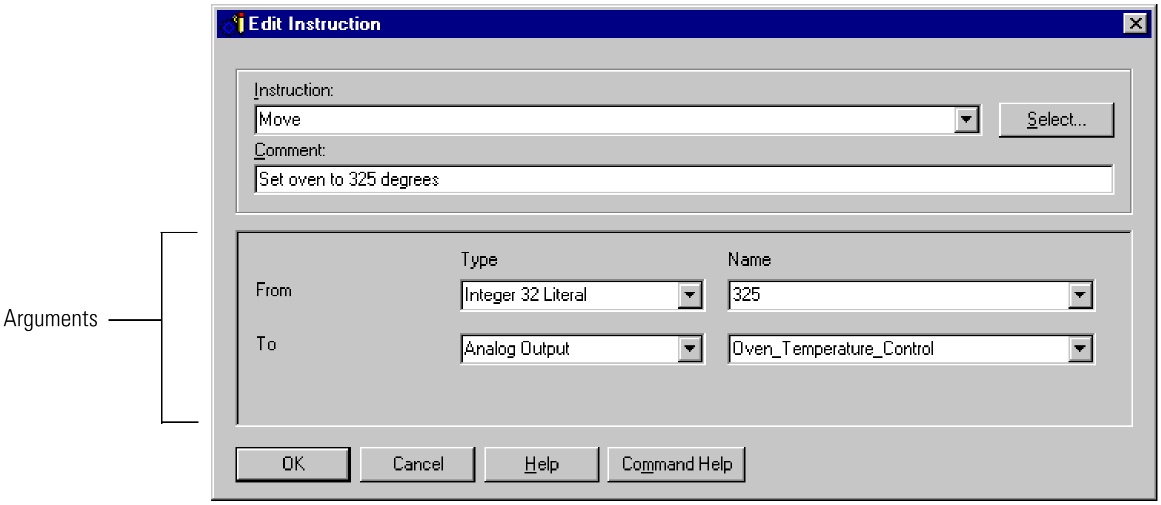
For more information on any command, click the command name and press F1 to open online help. You can also look up the command in the *ioControl Command Reference*.

6. When the command you want is highlighted, click OK to return to the Add Instruction dialog box.

7. (Optional) Enter a comment about the purpose of the instruction.

Comments help explain the command's purpose in this block and are helpful to anyone who debugs or updates the strategy later.

8. Complete each argument for the command by typing in the Type and Name fields or by choosing from the drop-down lists.



Instruction: Move

Comment: Set oven to 325 degrees

	Type	Name
From	Integer 32 Literal	325
To	Analog Output	Oven_Temperature_Control

If the argument type is a literal (or constant), you must type it in.

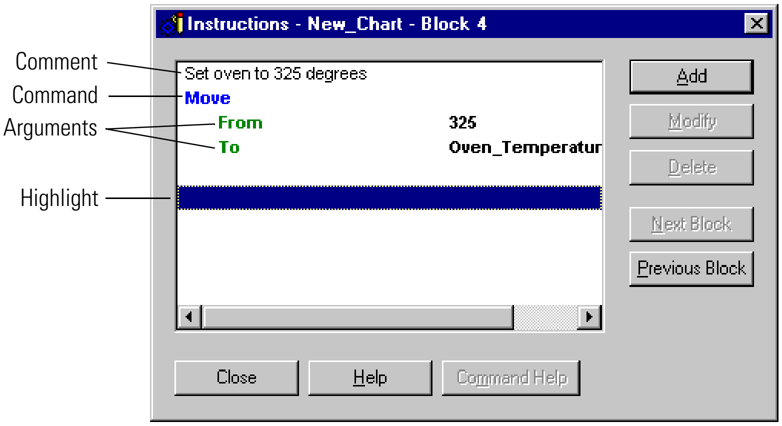
If you type in the name of an item that doesn't exist, for example a variable or I/O point, you are prompted to add it to the strategy.

Each command requires a certain number of arguments, from zero to eight. For help in completing arguments, see the *ioControl Command Reference* or the online help for the specific command you're using.

9. When the arguments are complete, click OK.

You return to the Instructions dialog box, which now shows the command you just added. Notice that the comment you entered appears just above the command. The arguments you entered appear as part of the instruction. The highlight shows where the next command will be placed if you add another command to this block.

Here is an example of an Instructions dialog box for an action block:



Instructions - New_Chart - Block 4

```
Set oven to 325 degrees
Move
    From                325
    To                  Oven_Temperatur
```

10. To add another command, click to place the highlight where you want the command to appear. Click Add and repeat the steps in this section.

If you highlight a command that's already in the dialog box, the new command will appear just before it.

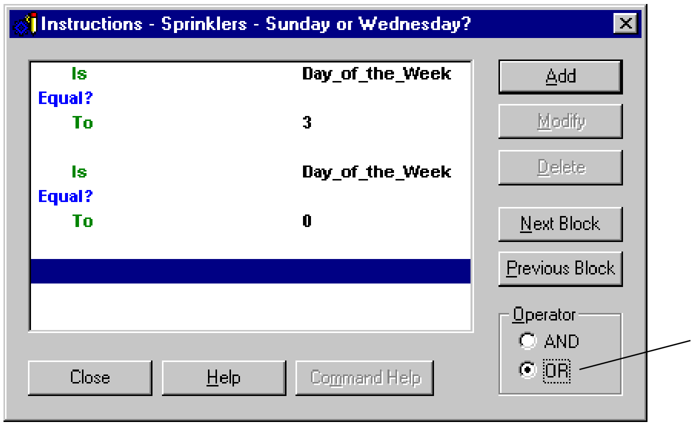
TIP: If you are adding commands to several blocks at once, you can quickly move from block to block in the chart by clicking the Next Block or Previous Block buttons in the Instructions dialog box.

If you're working with a condition block, clicking Next Block opens a dialog box so you can select which block to choose, since condition blocks have two exits. The same dialog box appears if you click Previous Block and the block you're working with has connections coming from more than one block.

11. If you put more than one command in a condition block, complete the Operator group as follows:

- If both commands must be true to exit the block true, click AND.
- If only one of the commands must be true to exit the block true, click OR.

Here is an example of an Instructions dialog box for a condition block with two commands. In this case, the block will exit true if either of the commands is true.



```
    Is          Day_of_the_Week
Equal?
    To          3

    Is          Day_of_the_Week
Equal?
    To          0
```

Either the first command OR the second command can be true for this condition block to exit true.

Changing a Command

1. With the strategy open in Configure or Online mode and the flowchart open, double-click the block containing the command you want to change.

   NOTE: To change commands in OptoScript Blocks*, see "Using the OptoScript Editor" on page 11-25.*

2. In the Instructions dialog box, double-click any line of the command you want to change.

   You can also click the command to highlight it and click Modify.

3. In the Edit Instruction dialog box, make the necessary changes.

   For help, see "Adding Commands" on page 9-18.

4. Click OK to return to the Instructions dialog box, where you can see the changed command.

Deleting a Command

You can delete a command permanently, or you can comment out a command so it is temporarily skipped, usually for debugging purposes.

Permanently Deleting a Command

1. With the strategy in Configure or Online mode and the flowchart open, double-click the block containing the command you want to delete.

NOTE: To delete commands in OptoScript Blocks, see "Using the OptoScript Editor" on page 11-25.

2. In the Instructions dialog box, click any line of the command you want to delete.

   CAUTION: *Make sure you select the correct command. You cannot undo a deletion!*

3. Click Delete or press DELETE on the keyboard.

Commenting Out a Command

You can mark certain commands so that the Strategy temporarily ignores them. Commenting out one or more commands can help you pinpoint problems in a strategy.

1. With the strategy in Configure or Online mode and the flowchart open, double-click the block containing the command(s) you want to comment out.

   NOTE: To comment out commands in OptoScript Blocks, see "Using the OptoScript Editor" on page 11-25.

2. In the Instructions dialog box, click the first command you want to comment out. Click Add.

3. In the Add Instructions dialog box, choose the instruction Comment (Block). Click OK.

   You return to the Instructions dialog box, and all the instructions in the block from that point on are grayed out, indicating that they will be ignored when the strategy runs.

4. Click just beyond the last command you want to comment out. Add another Comment (Block) instruction.

   When you return to the Instructions dialog box, all commands between the two Comment (Block) instructions are grayed out.

5. When you no longer want the strategy to ignore the command(s), delete the two Comment (Block) instructions.

   NOTE: The Comment (Block) command is used for this purpose. The Comment (Instruction) command just places an explanatory comment in the Instructions dialog box. It does not affect any commands.

Cutting or Copying a Command

For commands in action, condition, and continue blocks you can cut or copy commands to the Windows clipboard and then paste them in the same block or in another block, or even in a block of another chart within the same Strategy. For OptoScript Blocks, see "Using the OptoScript Editor" on page 11-25.

1. With the strategy in Configure or Online mode and the flowchart open, double-click the block containing the command you want to cut or copy.

2. In the Instructions dialog box, click any line of the command you want to cut or copy.

   To cut or copy more than one command, hold down the SHIFT key while you click all the commands to be cut or copied at once.

   NOTE: To cut and paste all commands in a block, copy and paste the entire block, and then change its name if necessary.

3. Press CTRL+X to cut the command(s) or CTRL+C to copy them.

   You can also click the right mouse button and choose Cut or Copy from the pop-up menu.

   The command is cut or copied to the Windows clipboard.

Pasting a Command

Once you have cut or copied a command, you can paste it into any block in the same Strategy.

Choose one of the following, and then press CTRL+V:

- To paste the command in the same block, click where you want to insert the command.
- To paste the command at the end of the instruction block, move the cursor just below the last instruction and click to highlight the empty space.
- To paste the command to another block, click Close to exit the current Instructions dialog box and double-click the block where you want to place the command. Click where you want to insert the command.

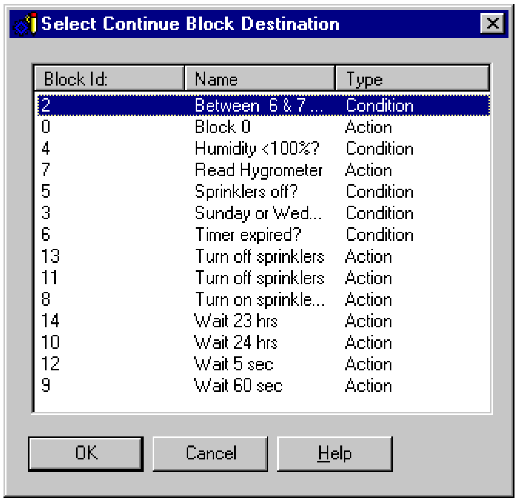
Configuring a Continue Block

Continue blocks do only one thing: jump to another block. Thus, the only information you need to provide in a continue block is the name of the block to jump to.

1. With the strategy in Configure or Online mode and the flowchart open, double-click the continue block you want to configure.

   You can also click the block, click the right mouse button, and choose Detail from the pop-up menu.

The Select Continue Block Destination dialog box appears, listing all blocks in the chart.



Block Id:	Name	Type
2	Between 6 & 7 ...	Condition
0	Block 0	Action
4	Humidity <100%?	Condition
7	Read Hygrometer	Action
5	Sprinklers off?	Condition
3	Sunday or Wed...	Condition
6	Timer expired?	Condition
13	Turn off sprinklers	Action
11	Turn off sprinklers	Action
8	Turn on sprinkle...	Action
14	Wait 23 hrs	Action
10	Wait 24 hrs	Action
12	Wait 5 sec	Action
9	Wait 60 sec	Action

2. Click the destination block and click OK.

Viewing and Printing Chart Instructions

To view or print commands in a chart, see "Viewing and Printing Strategy or Subroutine Commands" on page 7-25.

CHAPTER 10

Programming with Commands

Introduction

Commands (or instructions) in ioControl are roughly grouped by function. This chapter tells you what you need to know about each group in order to program your ioControl strategy effectively. For detailed information on using a command, see the *ioControl Command Reference*, where commands are listed alphabetically.

In this Chapter

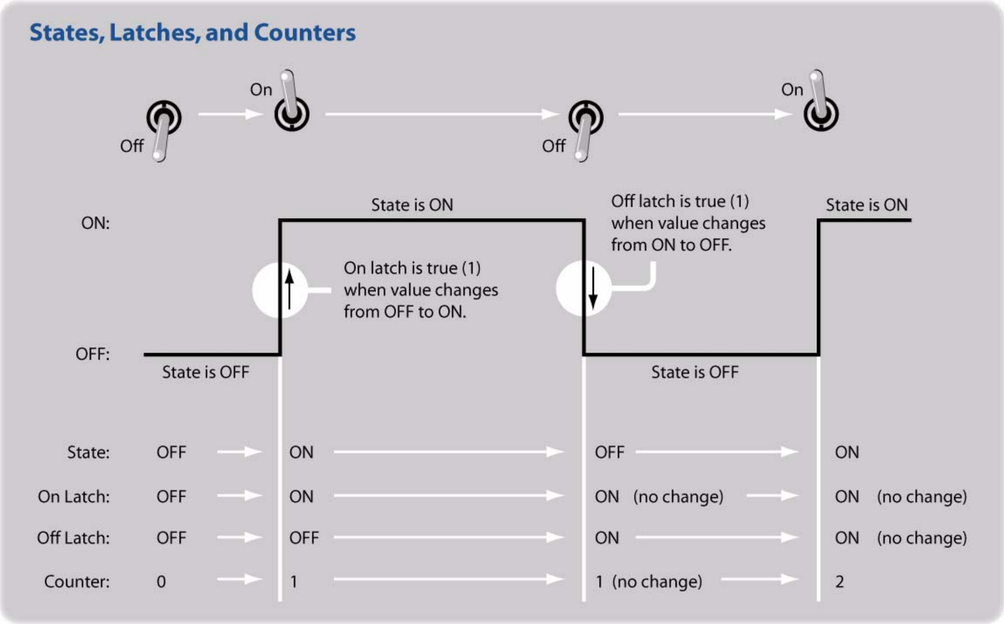
Digital Point Commands

The following commands are used with standard Digital Points only. For high-density digital points, see page 10-6.

Basic Commands Turn On Turn Off On? Off?	**Latches** Get Off-Latch Get On-Latch Clear Off-Latch Clear On-Latch Clear All Latches Get & Clear Off-Latch Get & Clear On-Latch Off-Latch Set? On-Latch Set?
Totalizers* Get Off-Time Totalizer Get On-Time Totalizer Get & Restart Off-Time Totalizer Get & Restart On-Time Totalizer	
Counters* Start Counter Stop Counter Get Counter Clear Counter Get & Clear Counter	**Pulses*** Generate N Pulses Start Off-Pulse Start On-Pulse Get Off-Pulse Measurement Get Off-Pulse Measurement Complete Status Get & Restart Off-Pulse Measurement Get On-Pulse Measurement Get On-Pulse Measurement Complete Status Get & Restart On-Pulse Measurement Start Continuous Square Wave
Period and Frequency* Get Frequency Get Period Get Period Measurement Complete Status Get & Restart Period Set TPO Percent Set TPO Period	
*Some digital point commands are available in ioControl Professional only. Some commands are available only on some I/O units. For details, see specific information for each command in the *ioControl Command Reference* or online Help.	

States, Latches, and Counters

The following diagram illustrates states, latches, and counters. While states and latches apply to Digital Points on all I/O units, counters depend on the capability of the brain. See the brain's data sheet for specifications.


States, Latches, and Counters
Off
On
Off
On
ON:
OFF:
State is ON
State is OFF
On latch is true (1) when value changes from OFF to ON.
Off latch is true (1) when value changes from ON to OFF.
State is OFF
State is ON

State:	OFF	ON	OFF	ON
On Latch:	OFF	ON	ON (no change)	ON (no change)
Off Latch:	OFF	OFF	ON	ON (no change)
Counter:	0	1	1 (no change)	2

Latches

Latches are an extremely high-speed digital function. Both on-latches and off-latches are available. Latches are automatic and do not have to be configured.

When the value of a digital Input Point changes from off to on, an on-latch is automatically set. While the value of the point may return to off, the on-latch remains set until cleared, as a record of the change. Similarly, an off-latch is set when the value of a digital point changes from on to off, and it remains set until cleared.

To read a latch and clear it at the same time, use the command Get & Clear On-Latch or Get & Clear Off-Latch.

Counters

Most standard digital inputs can be used as counters, to count the number of times the input changes from off to on. The availability of counters depends on the brain's capabilities, and the speed of counters depends on the module; see the brain's and module's data sheets for specifications.

Before using a counter, you must configure the point as a counter. (See "Adding a Digital I/O Point" on page 6-16 or use ioManager.) The counter begins counting as soon as it is configured. You do not need to use the Start Counter command to start it.

Quadrature Counters

Quadrature counters require a special module and configuration, but once they are configured, you use the same commands (such as Start Counter and Clear Counter) for them as for regular counters.

Be aware that quadrature counters differ on Ethernet-based and mistic I/O units. On mistic I/O units, quadrature counters must be started with the command Start Counter, and a positive value means that phase B leads phase A. On Ethernet-based I/O units, counters start as soon as they are configured, the Start Counter command is only used after Stop Counter, and a positive value means that phase A leads phase B. See additional details in the *ioControl Command Reference* or online Help.

Totalizers

Digital totalizers track the total time a specific Input Point has been on or off. For example, you could track how long a pump, fan, or motor has been on. Digital totalizers are useful for periodic maintenance. Before using a totalizer, you must configure the point with this feature. (See "Adding I/O Points" on page 6-16 for help.) The availability of totalizers depends on the brain; see the brain's data sheet for more information.

To check total time and leave the totalizer running, use Get Off-Time Totalizer or Get On-Time Totalizer. To check total time and reset the totalizer to zero, use Get & Restart Off-Time Totalizer or Get & Restart On-Time Totalizer.

Pulses

Pulsing commands send on- and off-pulses to an Output Point. The availability of pulsing depends on the brain; see the brain's data sheet for specifications.

Generate N Pulses. The command Generate N Pulses is frequently used to flash a light or sound an alarm. For example, you could sound a two-second alarm four times. In the arguments, you set the number of times the on-pulse is sent, the length of the on-pulse, and the length of the off-pulse. Generate N Pulses always starts with an off-pulse. If you resend this command, make sure to leave sufficient time in between so it does not interfere with itself.

Start On Pulse and Start Off Pulse. The commands Start On Pulse and Start Off Pulse send a single pulse cycle:

- Start On Pulse starts with an on-pulse of a length you determine, and ends with an off-pulse.
- Start Off Pulse starts with an off-pulse of a length you determine, and ends with an on-pulse.

Both of these commands can be used as time delays. For example, if a light is on and you want to turn it off after 30 seconds, you can send a Start On Pulse command, setting the on-pulse to be 30 seconds long. At the end of that time, the off-pulse is sent to turn off the light.

You can also use this type of command in a loop to turn a digital point on or off for short intervals. For example, you could create a loop that checks the level of liquid in a tank and pulses on a drain if the level is too high. The advantages of using a pulse command are that the point does not have to be turned off, and if communication is lost to the point, the point does not remain on.

Pulse Measurement commands measure pulses on digital input points. For details, see the specific command in the *ioControl Command Reference* or online Help.

IVAL and XVAL

All I/O points have two associated values: XVAL and IVAL. If you are using ioControl in Debug mode to manipulate I/O values or to disable an I/O point or I/O unit, you need to understand these values.

XVAL The external value, or XVAL, is the "real" or hardware value as seen by the I/O unit. This value is external to the control engine.

IVAL The internal value, or IVAL, is a logical or software copy of the XVAL that is in the control engine. The IVAL may or may not be current, since it is updated to match the XVAL only when a strategy in the control engine reads or writes to an I/O point.

Do not be concerned if the IVAL does not match the XVAL. A mismatch just means that the program is not reading from or writing to the I/O point in question at the moment.

Simulation and Test: The "Real" Use for XVAL and IVAL

To test output performance, you may want to force an XVAL for a specific output to a particular value. If the program is actively writing to the output, you need to disable the output to do so. If the program is stopped, there is no need to disable it.

To test program logic, you may want to force an IVAL for a specific input to a particular value. To do so, you must disable the input first.

You can disable an I/O point or unit in two ways. The more common way is from within Debug mode, by double-clicking a point on the Strategy Tree and modifying the point's settings and values through the Inspection dialog box. The second way is from within the strategy, using commands such as Disable Communication to Digital Point, Disable Communication to Analog Point, or Disable Communication to I/O Unit. (See "Simulation Commands" on page 10-69.)

Additional Commands to Use with Standard Digital Points

Although not listed under Digital Point commands, several other commands can be used for digital operations:

- Use **Move** to cause an output to assume the state of another input or output. A digital input or output that is on returns a True (non-zero). A True (non-zero) sent to a digital output turns it on.

- Use **NOT** to cause an output on one I/O unit to assume the opposite state of an input on another I/O unit.
- Use **Get I/O Unit as Binary Value** to get the state of all points at once. Then use **Bit Test** to determine the state of individual points. This method is much faster than reading each point individually.
- Use **Set I/O Unit From MOMO Masks** to control all outputs at once.

Standard Digital Points and OptoScript Code

In OptoScript code, a standard digital I/O point can be used directly, wherever a numeric variable can be used. For example, you can turn a point off by assigning it a value of zero, or turn it on by assigning it a non-zero value. You can also use standard Digital Points directly in mathematical expressions and control structures. For more information, see "Using I/O in OptoScript" on page 11-12.

High-Density Digital Module Commands

The following commands are used with SNAP high-density digital input and output modules:

States
Get HDD Module States
Get All HDD Module States
Set HDD Module from MOMO Masks
Turn On HDD Module Point
Turn Off HDD Module Point

Counters
Get HDD Module Counters
Get & Clear HDD Module Counter
Get & Clear HDD Module Counters

Latches
Get HDD Module On-Latches
Get HDD Module Off-Latches
Get All HDD Module On-Latches
Get All HDD Module Off-Latches
Clear HDD Module On-Latches
Clear HDD Module Off-Latches
Get & Clear HDD Module On-Latches
Get & Clear HDD Module Off-Latches
Get & Clear All HDD Module On-Latches
Get & Clear All HDD Module Off-Latches

About High-Density Digital Modules

SNAP high-density digital modules can be used on I/O units with the following brains:

SNAP-UP1-ADS
SNAP-UP1-M64
SNAP-B3000-ENET
SNAP-ENET-RTC
SNAP-ENET-S64

High-density modules cannot be used with digital-only brains, because they communicate with the brain as an analog or special-purpose module communicates.

Comparing SNAP High-Density and Standard Digital Modules

SNAP high-density digital modules are different in several other ways from standard SNAP digital input and output modules:

- Standard digital modules contain four points per module; high-density digital modules contain 32 points per module.
- The points on a standard digital input module are isolated from each other. Points on high-density input modules are in four groups of eight points; groups are isolated from each other, but points within a group are not isolated.
- Standard digital modules have LEDs for each point visible on the module's top; high-density modules do not have LEDs.
- Standard digital modules can be placed only in digital slots on the mounting rack; high-density digital modules can be placed anywhere, even in slots marked "Analog Only."
- The turn-on/turn-off time is faster for standard digital modules than for high-density modules. The update time (time for data to pass from the module to the brain) is also faster for standard digital modules and is determined by the speed of the module itself. In high-density modules, update time depends on the brain's analog scanner and is affected by the number of modules on the rack and how busy the brain is with Ethernet communication. (For more specific information on turn-on/turn-off and update times, see the data sheets for standard and high-density modules.) You may find that inserting Delay commands in your strategy provides more accurate results, especially with counters.
- **IMPORTANT:** Each point on a standard digital module is given a name when configured in ioControl or ioManager, and your strategy refers to the point by its name. Points on high-density modules do not have individual names. HDD modules do not require configuration, so their points do not appear in the Configure I/O Points dialog box nor on the Strategy Tree. Because HDD points do not have names, most ioControl commands use bitmasks to read or write to them.
- Points on HDD modules cannot be disabled in Debug mode for simulating inputs and outputs.
- Counting is done differently. See "Counting on High-Density Digital Modules" below for details.

Counting on High-Density Digital Modules

On standard SNAP digital input modules, any point can be configured as a counter because counting is done on the brain. We refer to it as "high-speed" counting because it can be very fast, depending on the speed of the module.

On high-density SNAP digital modules, however, the module itself does the counting, so no configuration is necessary. The module uses a 16-bit counter, but the brain used with the module accumulates counts to 32 bits by periodically getting and clearing the module's counts and adding each new count to what it already has for each point. Update time varies based on the number of modules on the rack and Ethernet communication demands on the brain. When using ioControl's signed 32-bit variables, one bit of the 32 is used for the integer's sign (+/–), so counts over 2 billion may not be accurate.

Because counting is done in the module rather than in the brain, you can get counts for high-density digital modules used with SNAP-UP1-M64 and SNAP-ENET-S64 brains, which do not support high-speed counting.

Counting on HDD modules is at 0–50 Hz at a 50% duty cycle. This rate is useful for applications that require counting but not at high speeds—for example, rotating shafts, flow meters that generate pulses, and electrical meters tuned to slower speeds.

Using HDD Module Counters

Counters on HDD modules are always active; you do not need to configure a counter before using it. Because counters are always active, you should clear a counter before using it to make sure the count accurately reflects your current task.

Counters are returned as 32-bit integers, either in a variable for one point, or in an integer 32 table for all points on a module. The table must contain 32 elements beyond the starting index, since there are 32 points on the module.

See the *ioControl Command Reference* or ioControl command Help for more information about specific commands.

Using HDD Module Commands

As shown in the following table, HDD module commands work either with an individual point, with all points on one module, or with all HDD modules on the I/O unit.

Purpose	Command name	Reads or writes to:		
		One point	One module	All modules
Reading states and latches and clearing latches	Clear HDD Module Off-Latches		X	
	Clear HDD Module On-Latches		X	
	Get & Clear HDD Module Off-Latches		X	
	Get & Clear HDD Module On-Latches		X	
	Get & Clear All HDD Module Off-latches			X
	Get & Clear All HDD Module On-latches			X
	Get All HDD Module States			X
	Get All HDD Module Off-Latches			X
	Get All HDD Module On-Latches			X
	Get HDD Module States		X	
	Get HDD Module Off-Latches		X	
	Get HDD Module On-Latches		X	
Reading and clearing counters	Get HDD Module Counters		X	
	Get & Clear HDD Module Counters		X	
	Get & Clear HDD Module Counter	X		

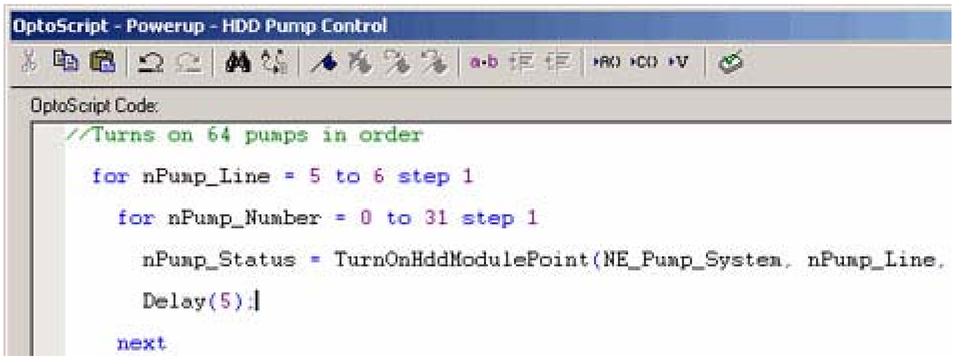
Purpose	Command name	Reads or writes to:		
		One point	One module	All modules
Writing to points	Set HDD Module from MOMO masks		X	
	Turn On HDD Module Point	X		
	Turn Off HDD Module Point	X		

Individual Point

A few HDD module commands, such as Turn On HDD Module Point and Get & Clear HDD Module Counter, address an individual point by module and point number. Remember that Opto 22 modules and points start at zero.

HDD points do not have names. If you need to refer to a specific point or a group of points by name, however, you can create an ioControl variable and place data in it.

Creating a variable for a group of points can be very useful in a large installation with many similar inputs or outputs. For example, suppose output modules 5 and 6 on the NE_Pump_System I/O unit each control 32 pumps. You could use an integer 32 variable called nPump_Line for the module number and another integer 32 variable called nPump_Number for the point number. The variables make it easy to loop through all pumps to turn them on, as shown in the following OptoScript example.



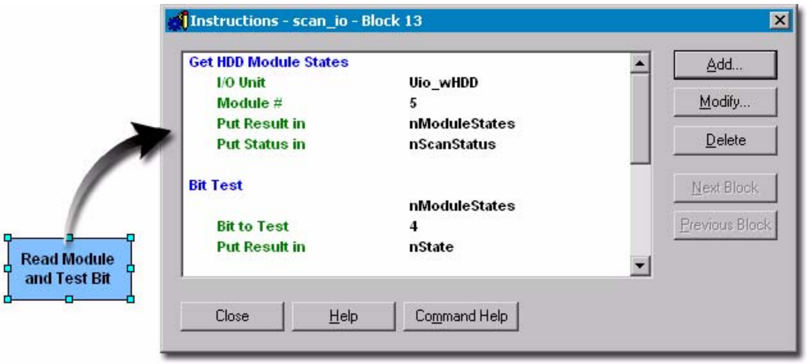
```
//Turns on 64 pumps in order
 for nPump_Line = 5 to 6 step 1
  for nPump_Number = 0 to 31 step 1
   nPump_Status = TurnOnHddModulePoint(NE_Pump_System, nPump_Line,
   Delay(5);
  next
```

All Points on a Module

Several commands, such as Get HDD Module States, work with all points on a module at once by using a bitmask. The least significant bit corresponds to point zero. A bit with a value of 1 indicates that the corresponding point is on; a bit with a value of 0 indicates the point is off.

Because HDD points on a module are represented and controlled through bitmasks, Logical Commands such as Bit Test and Bit Clear are useful for manipulating the bits.

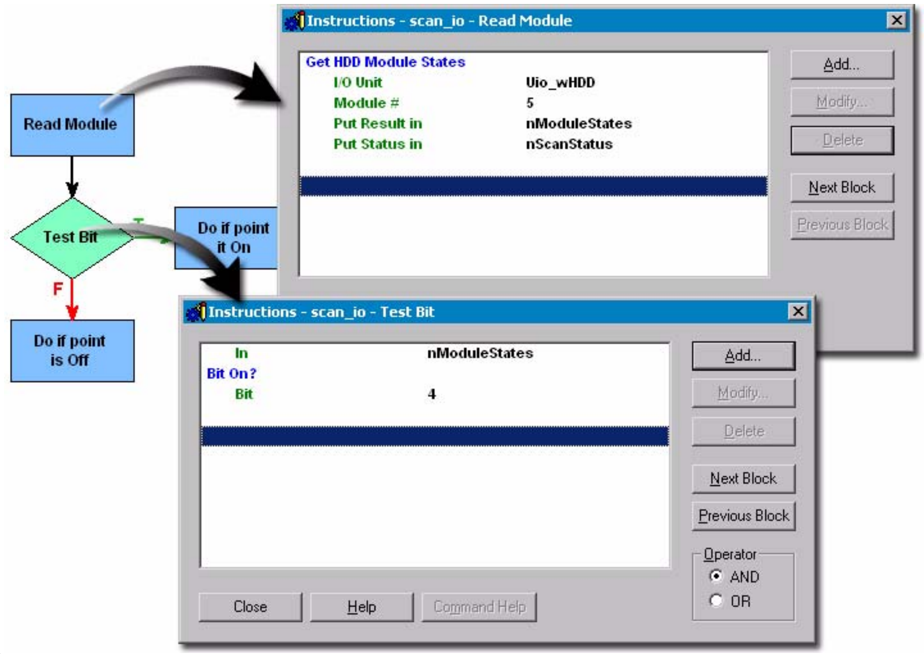
The following example reads the states of all points on the module in rack position 5 and places the bitmask in an integer variable, nModuleStates. To read point 4, which corresponds to bit 4, you can use Bit Test. The result of the bit test is placed in the variable nState. If nState = 1, the point is on; if nState = 0, the point is off.



Instructions - scan_io - Block 13

```
Get HDD Module States
    I/O Unit            Uio_wHDD
    Module #            5
    Put Result in       nModuleStates
    Put Status in       nScanStatus

Bit Test
                        nModuleStates
    Bit to Test         4
    Put Result in       nState
```

You can place the bit's value in a variable and test it as in the example above, or you can use the bit to direct flowchart logic. In the example below, the Bit On? command tests the bit and exits true if the bit is 1, false if the bit is 0.



Instructions - scan_io - Read Module

```
Get HDD Module States
    I/O Unit            Uio_wHDD
    Module #            5
    Put Result in       nModuleStates
    Put Status in       nScanStatus
```

Instructions - scan_io - Test Bit

```
    In                  nModuleStates
Bit On?
    Bit                 4
```

Operator: ● AND ○ OR

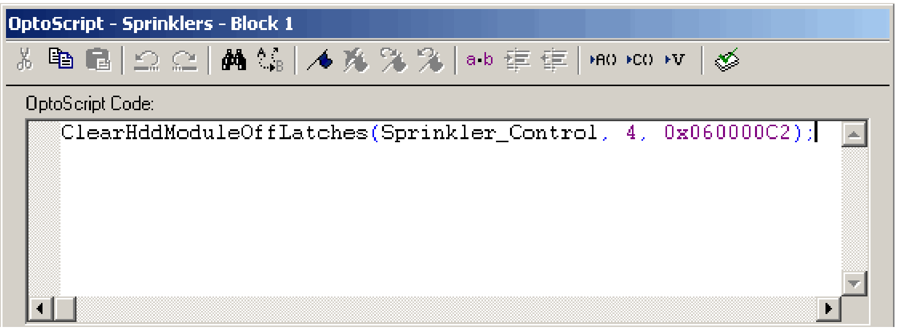
Here's another example of a command that works with all points on the module at once. Suppose you want to reset the off-latches on a few points on the module in rack position 4—specifically, points 1, 6, 7, 25, and 26. Using Clear HDD Module Off-Latches, you indicate the I/O unit, the module number (4), and then use a bitmask to determine the off-latches to clear. The bitmask would be:

```
00000110000000000000000011000010
```

As you can see in the illustration below, the 1 bits in the mask will affect their respective points, so off-latches for point numbers 1, 6, 7, 25, and 26 will be cleared. (To save space, only the first 8 and last 8 off-latches are shown.)

	Point Number	31	30	29	28	27	26	25	24	→	7	6	5	4	3	2	1	0
Bit Mask	Binary	0	0	0	0	0	1	1	0	→	1	1	0	0	0	0	1	0
	Hex	0				6				→	C				2			

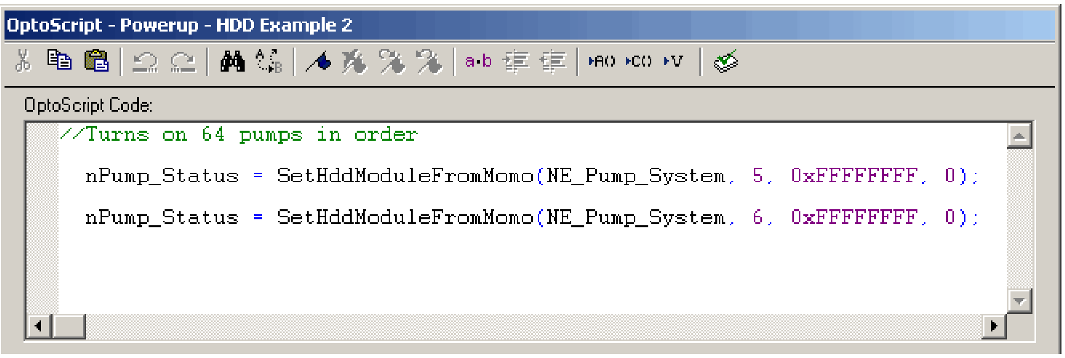
Here is this example shown in OptoScript code:



```
ClearHddModuleOffLatches(Sprinkler_Control, 4, 0x060000C2);
```

NOTE: In standard ioControl commands, you can enter the bitmask in binary or in hex, depending on the integer display you've chosen under the View menu. In OptoScript code, use hex. In either case, don't use decimal to represent masks, since ioControl's 32-bit integers use the most significant digits as the sign (+/–).

As a final example of working with all points on a module at once, here is a rewritten version of the pump control script shown on page 10-9. In this example, the pumps controlled by HDD Output modules 5 and 6 on the I/O unit NE_Pump_System are turned on by the Set HDD Module From MOMO Masks command. Notice that this command turns on all the pumps controlled by a single module simultaneously, rather than one after the other.



```
//Turns on 64 pumps in order
  nPump_Status = SetHddModuleFromMomo(NE_Pump_System, 5, 0xFFFFFFFF, 0);
  nPump_Status = SetHddModuleFromMomo(NE_Pump_System, 6, 0xFFFFFFFF, 0);
```

All HDD Modules on the I/O Unit

Commands with the word “All” in their title—such as Get All HDD Module States—work with all HDD modules on one I/O unit at once. They do so using a table; each element in the table contains a bitmask representing the data for one module.

For example, to read the states of all points on all HDD modules on one I/O unit, you would use the command Get All HDD Module States. If the I/O unit consists of an 8-module rack filled with HDD modules, the table would contain data such as the following.

Index	Value (Bitmask)
0	00000001000101000000000110010000
1	01100001010001110000001010110010
2	00000000000010000100010000000111
3	00100000011000000010010001000100
4	01100001010001110000001010110010
5	00001110000010000110010000001001
6	10000000110000001110000000100100
7	00110000011100001111100000000001
8	00000000000000000000000000000000
↓	↓
15	00000000000000000000000000000000

Each index contains the status data for the HDD module in the corresponding position on the rack. A value of 1 indicates that the point is on; a value of 0 indicates that it is off. The least significant bit in the mask corresponds to point zero on the module.

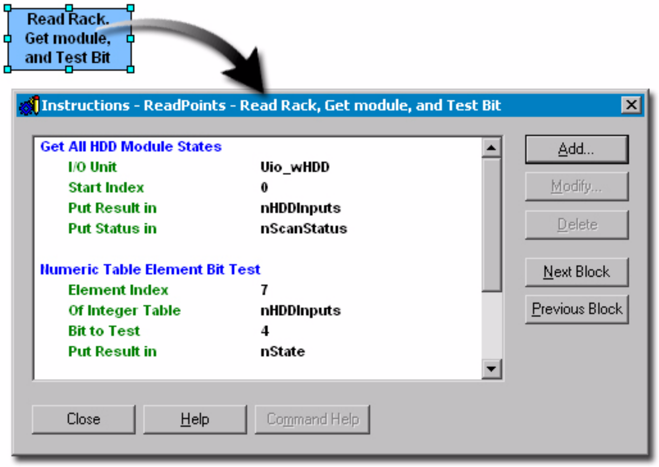
In this example, index 2, which contains the status of all points on the module in slot 2, shows that points 0, 1, 2, 10, 14, and 19 are on. All other points on the module are off.

The remainder of the table is zero-filled, since there are no more modules.

Returned data is only for HDD modules. If the rack contained a standard digital module in position 3 and a serial module in position 4, elements 3 and 4 in the table would be zero-filled.

To use the data in the table, you can manipulate the table using commands such as Shift Numeric Table Elements and Numeric Table Element Bit Set.

For example, the first command in the block shown below places data for all modules on the rack in the table nHDDInputs, starting at table element zero. The second command tests bit 4 in table element 7 (which corresponds to point 4 on the module in position 7) and puts the result of the test into the variable nState.



Instructions - ReadPoints - Read Rack, Get module, and Test Bit

Get All HDD Module States

I/O Unit	Uio_wHDD
Start Index	0
Put Result in	nHDDInputs
Put Status in	nScanStatus

Numeric Table Element Bit Test

Element Index	7
Of Integer Table	nHDDInputs
Bit to Test	4
Put Result in	nState

If you need to work with the data for only one module in the table, you can use the command Move From Numeric Table Element to move the module's data from the table into an integer variable. Then you can use bit commands within the integer.

Analog Point Commands

The following commands are used with Analog Points:

Offset and Gain
Set Analog Offset
Calculate & Set Analog Offset
Set Analog Gain
Calculate & Set Analog Gain

Minimum/Maximum Values
Get Analog Minimum Value
Get & Clear Analog Minimum Value
Get Analog Maximum Value
Get & Clear Analog Maximum Value

Others
Ramp Analog Output[1]
Set Analog TPO Period
Set Analog Load Cell Fast Settle Level
Set Analog Load Cell Filter Weight
Set Analog Filter Weight
Set Analog Totalizer Rate[1,2]
Get Analog Totalizer Value[1,2]
Get & Clear Analog Totalizer Value[1,2]
Get Analog Square Root Value[1,2]
Get Analog Square Root Filtered Value[1,2]
Get Analog Filtered Value[1,2]
Get & Clear Analog Filtered Value[1,2]

[1] Applies to ioControl Professional only
[2] Applies to mistic I/O units only

Offset and Gain Commands

The easiest way to set offset and gain is to do so when you configure Analog Points in ioManager, using the Calibrate button in the Configure I/O Points dialog box.

You can also set offset and gain in ioControl. If you already know the offset and gain for a point, you can use the commands Set Analog Offset and Set Analog Gain. If you do not know the offset and gain, you can use the commands Calculate & Set Analog Offset and Calculate & Set Analog Gain to have the brain calculate them. Calculate offset first, and then calculate gain.

By setting offset and gain, you make sure that values read are accurate.

Offset is the difference between the minimum input of an analog Input Point and the actual minimum signal received from a field device. For example, if a 4–20 mA input receives a minimum signal that is slightly off (not exactly 4 mA), the difference between the two minimums is the offset. Reading ± Offset = Actual Value. For example:

If minimum input =	4.000 mA
and zero-scale reading =	4.003 mA
then offset =	-0.003 mA

Gain is the difference in the full-scale reading, but expressed differently.
Measured Value ∗ Gain = Actual Value. For example:

If maximum input =	20.00 mA
and measured value =	20.50 mA
then gain =	0.9756097560976

Minimum/Maximum Values

The Opto 22 brain automatically keeps track of minimum and maximum values for analog Input Points. Min/max values are often used to monitor pressure or temperature.

To read the minimum or maximum value and leave it as is, use Get Analog Minimum Value or Get Analog Maximum Value. To read the minimum or maximum value and clear it—for example, to record the minimum pressure in each 24-hour period—use Get & Clear Analog Minimum Value or Get & Clear Analog Maximum Value.

Analog Totalizers

Analog totalizers are available on mistic I/O units only.

Analog totalizers are used to track total volume or quantity. For example, if an Analog Point measures gallons per minute, you could use an analog totalizer to determine the total number of gallons moved over a period of time.

To read the value and leave the totalizer running, use the command Get Analog Totalizer Value. To read the value and set the totalizer back to zero, use the command Get & Clear Analog Totalizer Value.

Analog Points and OptoScript Code

In OptoScript code, an analog I/O point can be used directly, wherever a float variable can be used. For example, you can assign an Analog Point a value, or use points directly in mathematical expressions and control structures. For more information, see "Using I/O in OptoScript" on page 11-12.

I/O Unit Commands

The following commands are used to communicate with an I/O unit, which controls a group of I/O points:

I/O Unit Ready?
Set I/O Unit from MOMO Masks[1,2]

Get I/O Unit as Binary Value
IVAL Move Numeric Table to I/O Unit
Move I/O Unit to Numeric Table
Move Numeric Table to I/O Unit
Write I/O Unit Configuration to EEPROM
Get Target Address State[1]
Set Target Address State[1]
Set All Target Address States[1]

[1] ioControl Professional only
[2] mistic I/O units only

CAUTION: *Write I/O Unit Configuration to EEPROM is not the recommended method for saving configuration to flash memory. If it is used too often or is in a loop within a strategy, flash memory can literally wear out. Instead of using this command in the strategy, it is better to store configurations to flash using ioManager (see the* ioManager User's Guide *for instructions) or using ioControl in Debug mode (see page 6-52).*

Commands for Ethernet Link Redundancy

The three target address commands (Get Target Address State, Set Target Address State, and Set All Target Address States) are used to manually change the path of communication between the controller and the I/O unit(s), based on the IP address used for the I/O unit. These commands let you switch communication from a primary to a secondary IP address (or vice versa) or enable or disable the primary or secondary address.

Ethernet link redundancy to I/O units is available only in ioControl Professional, only from a SNAP PAC controller, and only to Ethernet-based I/O units. The secondary IP address for an I/O unit may be for the second Ethernet network interface on a SNAP PAC R-series controller, or it may be for a separate I/O unit. If it is a separate unit, the primary and secondary I/O units must be the same type (for example, SNAP-ENET-S64) and have exactly the same points, because they are configured together under one name.

One or both target addresses (primary and secondary) can be enabled, but only one of them is active at any time. For link redundancy, both must be enabled. When an I/O unit is configured with two IP addresses, the default is for both to be enabled and the primary address to be active. If communication fails through the primary address, the control engine automatically switches to the secondary address. It continues to use the secondary address until communication fails through the secondary address or until you change the active address using Set Target Address State (for one I/O unit) or Set All Target Address States (for all I/O units on the control engine).

You may also want to use these commands to disable one address, for example if you are doing maintenance or repair on a network segment and need to switch communication to another segment temporarily. Disabling one address, of course, means that you no longer have link redundancy. If both addresses are disabled or unavailable, then communication is not possible and the I/O unit becomes disabled.

You can find out which addresses are enabled for an I/O unit and which address is currently active by using Get Target Address State.

To use these commands, you must have already designated primary and secondary IP addresses when configuring I/O units. See page 6-12 for steps. For additional information about link redundancy, see “Using Ethernet Link Redundancy in ioControl” on page 5-6.

Table Commands

The table commands for I/O units affect the states or values of all points on the I/O unit at once. For example, you can use the command Move I/O Unit to Numeric Table to read the states of all Digital Points and the values of all Analog Points on one I/O unit and place them into a table for easy retrieval. Table commands move data very quickly for faster throughput.

Other commands relating to tables can be found in “Miscellaneous Commands” on page 10-21, “Logical Commands” on page 10-33, “Mathematical Commands” on page 10-32, “Pointer Commands” on page 10-56, and “String Commands” on page 10-23.

Control Engine Commands

The following commands refer to the control engine:

Get Control Engine Address
Get Control Engine Type
Get Available File Space
Calculate Strategy CRC
Retrieve Strategy CRC

Get Firmware Version
Save Files To Permanent Storage
Erase Files In Permanent Storage
Load Files From Permanent Storage

Commands Relating to Permanent Storage

The term “Permanent Storage” in the last three commands listed above refers to the control engine’s flash memory. Files that are saved to flash memory remain in the control engine even when power to it is turned off.

These commands do NOT affect firmware files, configuration data, or strategy files saved to flash; they affect only files at the root of the control engine’s file system. For more information on the file system, see “Using the Control Engine’s File System” on page 10-42. For the specifics on individual commands, see online help or the *ioControl Command Reference*.

CAUTION: *Since these commands write to flash memory, use them sparingly within your strategy and make sure they do not end up in a loop. You can literally wear out flash memory if you save to it or erase it too many times.*

Chart Commands

The following commands control charts in the strategy:

Chart Running?
Chart Stopped?
Chart Suspended?
Get Chart Status

Calling Chart Running?
Calling Chart Stopped?
Calling Chart Suspended?
Continue Calling Chart

For information about charts in an ioControl strategy, see “ioControl Terminology” on page 3-6.

About the Task Queue

How do subroutines fit into the task queue? Whenever a chart calls a subroutine, the subroutine temporarily inherits the task in use by the calling chart along with its priority.

Does a task always use all of its allocated time? Not always. If a chart or subroutine runs in a loop, all allocated time is used. If a chart or subroutine does not need all of its allocated time to complete its job, all remaining time (including any portion of a time slice) is given up.

The following conditions cause a chart to use less than a full time slice:

- The chart or subroutine stops.
- The chart or subroutine is suspended.
- A Delay command is used.

Using the command Delay (mSec) with a value of 1 millisecond is a handy way to give up the time slice while waiting in a loop for a condition to become True. For more information, see “Increasing Efficiencies in Your Strategy” on page 4-22.

When does the requested change to a chart or task status take effect? Not immediately. In any multitasking system, timing and synchronization issues are always a concern. The time required for a particular request to be implemented depends on the number of tasks currently running and the specified chart's location in the task queue. We recommend using commands such as Calling Chart Suspended? and Chart Running? to determine the status of a chart.

Time/Date Commands

The following commands refer to time, dates, and days:

Dates and Days

Get Year
Set Year
Get Month
Set Month
Get Day
Set Day
Get Day of Week
Get Julian Day
Set Date
Copy Date to String (MM/DD/YYYY)
Copy Date to String (DD/MM/YYYY)

Time

Get Hours
Set Hours
Get Minutes
Set Minutes
Get Seconds
Get Seconds Since Midnight
Set Seconds
Set Time
Get System Time
Copy Time to String

These commands can be used for timing a process or for making sure things happen according to a set schedule. For example, you could use the command Get Seconds Since Midnight at the beginning of a process and again at the end of the process, and then subtract the two numbers to find out how long the process took.

You can set the time and date on the control engine by synchronizing it with the PC; in ioControl Debug mode, choose this option while viewing the control engine (Control Engine➞Inspect). You can also use these commands to set the time and date on the control engine.

Timing Commands

The following commands are used for timers and delays in a strategy.

Timers

Start Timer
Stop Timer
Pause Timer
Continue Timer
Timer Expired?
Set Down Timer Preset Value
Down Timer Expired?
Set Up Timer Target Value
Up Timer Target Time Reached?

Delays

Delay (mSec)
Delay (Sec)

Delay Commands

Delay commands are used frequently in strategies to pause the logic. Here are two reasons to use Delay (mSec) or Delay (Sec):

- To allow time for the state of an input to change before it is checked again. For example, a delay could give an operator time to release a button before the state of the button is rechecked, or allow time for an alarm state to change before rechecking.
- To let a chart give up the remainder of its time slice, when its logic does not need to run constantly. For more information on using delays in this way, see , see “Increasing Efficiencies in Your Strategy” on page 4-22..

Using Timers

Timers are a special type of numeric variable. An ioControl timer stores elapsed time in units of seconds with resolution of milliseconds. Down timers continuously count down to zero, and up timers continuously count up from zero. Timers can be paused and continued.

To create a timer in ioControl, configure a numeric variable and select the type Up Timer or Down Timer. You can use any ioControl command (for example, Move) that references a numeric variable to access a timer. You can view the current value of a timer at any time in ioControl Debug mode.

Since the timer is independent from the control engine’s clock, over thousands of seconds, the timer and the control engine’s clock will not match. Timers do not place any additional load on the CPU.

Down Timer Operation

The Set Down Timer Preset Value command sets the time the down timer will start from, but does not start the timer. Use the Start Timer command to start the timer counting down to zero. (Since the default preset value for a down timer is zero, nothing will happen if you use the Start Timer command before setting a value.)

Alternatively, you can use the Move command to set the time the down timer will start from. If you use Move, the down timer begins counting down immediately. If program execution speed is a priority, use the Move command and put an integer value rather than a float into the timer. This action eliminates the float-to-integer conversion time.

Note that if you use the Move command, any value you set using Set Down Timer Preset Value is overwritten, and subsequent Start Timer commands start the timer from the value last sent by the Move command.

To determine if the timer is finished, use the condition Down Timer Expired? This condition is true any time the down timer has a value of zero. Down Timer Expired? is much faster than using the condition Equal? to compare the timer to a value of zero.

The Stop Timer command forces the timer to stop *and puts its value at zero*. If you want to halt the timer and have it maintain its value at the time it was stopped, use the Pause Timer command instead. When you use Pause Timer, you can move the timer's value at the time it was stopped to a variable. You can also use the Continue Timer command to resume the timer where it left off.

Up Timer Operation

The Set Up Timer Target Value command sets the time for the Up Timer Target Time Reached? condition. It does not start the timer, however, and the timer does not stop when it reaches the target value. You must start the up timer from zero by using the Start Timer command.

If you use the Move command to move a value to an up timer, the value you moved becomes the target value and the up timer starts timing immediately. (Note that the timer does not start from the value you moved; it always starts at zero.)

The up timer does not stop when it reaches the target value. To determine if the timer has reached its target value, use the condition Up Timer Target Time Reached? This condition tests the timer to see if it is greater than or equal to the target time.

The Stop Timer command forces the timer to stop *and resets it to zero*. If you want to halt the timer and have it maintain its value at the time it was stopped, use the Pause Timer command instead. After you use Pause Timer, you can then move the timer's value at the time it was stopped to a variable. You can also use the Continue Timer command to resume the timer where it left off.

Miscellaneous Commands

The following commands are used with tables and for other purposes. Many of them are commonly used.

Comment (Block)
Comment (Single Line)
Float Valid?
Move
Move to Numeric Table Element
Move to Numeric Table Elements
Move from Numeric Table Element
Move Numeric Table Element to Numeric Table
Move Numeric Table to Numeric Table
Get Length of Table
Shift Numeric Table Elements
Generate Reverse CRC-16 on Table (32 bit)
Get Type From Name
Get Value From Name

Comment Commands

The comment commands listed above are used with standard ioControl commands in Action Blocks and Condition Blocks. For information on using comments in OptoScript Blocks, see page 11-28.

Comment (Single Line) and Comment (Block) commands are used in two entirely different ways:

- **Comment (Single Line)** enters a comment to help explain a block or an instruction within a block. Usually block names and comments within instructions are sufficient, but you can use Comment (Single Line) if you need more room for explanations.
- **Comment (Block)** comments *out* instructions. In other words, it tells the strategy to temporarily ignore certain instructions within a block. It can be useful in debugging or for saving work when a strategy is temporarily changed.

  To use it, place one Comment (Block) command at the beginning of the area you want to ignore, *and place another Comment (Block) command at the end of the area.* If you do not place the second Comment (Block) command, all the remaining instructions in that block are ignored. Areas that are commented out appear in the Instructions dialog box as gray.

Event Reaction Commands

The following commands are used with event/reactions on mistic I/O units only. They cannot be used with Ethernet-based I/O units.

NOTE: Similar events and reactions can be configured on a SNAP Ethernet-based I/O unit using ioManager, but they can interfere with ioControl strategy logic unless you are very careful. For more information, see the ioManager User's Guide (Opto 22 form #1440).

Event Occurred?
Event Occurring?
Get Event Latches
Get & Clear Event Latches
Clear Event Latch
Clear All Event Latches
Read Event/Reaction Hold Buffer

Disable Scanning for Event
Enable Scanning for Event
Disable Scanning for All Events
Enable Scanning for All Events
Disable Scanning of Event/Reaction Group
Enable Scanning of Event/Reaction Group
Event Scanning Disabled?
Event Scanning Enabled?

Understanding Event/Reactions (mistic I/O Units Only)

An event/reaction lets you distribute control logic to an I/O unit, so that some of the logic in a strategy can be run on the I/O unit independently of the controller. Event/reactions are supported by most mistic protocol brains. To verify, check the data sheet for the brain you are using.

As the name suggests, an event/reaction consists of an event and a corresponding reaction. The event is a state you define that the I/O unit can recognize. The defined state can be a combination of values, inputs, and outputs. Each time the event becomes true, its corresponding reaction is executed once.

On a digital multifunction I/O unit, for example, any pattern of input and output states (on and off) can constitute an event. On an analog I/O unit, an event could occur when an input channel attains a reading greater than a selected value. Examples of reactions include turning on or off a set of outputs, ramping an analog output, or enabling or disabling other event/reactions.

Event/reactions are stored in each I/O unit. As soon as power is applied to the I/O unit, all event/reactions for which scanning is enabled are scanned continuously in the same order in which they were configured in ioControl. Since each mistic I/O unit can be configured with up to 256 event/reactions, complex tasks and sequences can be performed on an I/O unit.

For step-by-step instructions on configuring event/reactions, see page 6-42.

Why Use Event/Reactions?

- To reduce communication overhead between the I/O unit and the controller.
- To distribute control logic sequences to the I/O unit rather than concentrating them in the controller.
- To handle high-speed logic functions more efficiently by distributing them to an I/O unit.
- To increase the execution speed of a strategy in the controller.

Typical Applications for Event/Reactions

- Motor-starting logic
- Drum sequencers
- Alarms
- Analog biasing
- Power-up sequencing

String Commands

The following commands are used with strings:

Move String
Move to String Table Element
Move to String Table Elements
Move from String Table Element
Compare Strings
String Equal?
String Equal to String Table Element?
Test Equal Strings
Get String Length
Append Character to String
Append String to String
Convert Float to String
Convert Number to String
Convert Number to String Field
Convert Hex String to Number
Convert Mistic I/O Hex to Float*
Convert IEEE Hex String to Number
Convert Number to Hex String
Convert Number to Mistic I/O Hex*
Convert Number to Formatted Hex String
Convert Integer 32 to IP Address String
Convert IP Address String to Integer 32
Convert String to Float
Convert String to Integer 32
Convert String to Integer 64
Convert String to Lower Case
Convert String to Upper Case
Find Character in String
Get Nth Character
Set Nth Character
Find Substring in String
Get Substring
Generate Checksum on String
Verify Checksum on String
Generate Forward CCITT on String
Verify Forward CCITT on String
Generate Reverse CCITT on String
Verify Reverse CCITT on String
Generate Forward CRC-16 on String
Verify Forward CRC-16 on String
Generate Reverse CRC-16 on String
Verify Reverse CRC-16 on String

* mistic I/O units only

Using Strings

NOTE: All numbers in this discussion of strings are decimal unless otherwise stated.

An ioControl string is a sequence of characters that can be grouped together. Characteristics of strings include the following:

- Strings are always referred to by name (and, if in a table, by index).

- Each character is represented by one byte.
- Each character is represented by its ASCII code (0 to 255).
- A string containing no characters is referred to as an *empty string*.
- Strings are frequently used in serial communication as a container for moving numeric characters from one device to another.
- Although a string may appear to contain numeric values, it does not. Digits “0” through “9” are characters just as much as “A” through “Z”; they do not represent numeric values.

  To illustrate, let’s look at the number 22. This is a decimal number representing a quantity of 22. The number 22 can be represented in a string in several ways; here are two of them:

  – As “22”: two character 50’s (The ASCII code for 2 is 50.)

  – As “16”: a character 49 (“1”) and a character 54 (“6”) (The hex value of 22 is 16.)

  Note that the string representation of the number 22 is no longer a number. It is simply one or two ASCII characters. The string representation of a number must be converted to a numeric value if it is to be used in calculations. Several Convert commands are available for this purpose.
- In standard ioControl commands, do not use double quotes around string literals. You can use single quotes, but they are not required.
- In OptoScript code, you must use double quotes for string literals. See Chapter 11, “Using OptoScript” for more information.

String Length and Width

The width of a string is the maximum length a string can be; length is the actual number of characters contained in the string. A string with a width of 100 may currently be empty, which means its length is zero. A string with a width of 10 containing the characters “Hello ” has a length of six (five for “Hello” and one for the space after the “o”). Although a string’s length may change dynamically as the string is modified by the program, its width remains constant.

When you configure a string variable or string table, you set the width of the string. All the strings in an ioControl string table must be of the same width.

ioControl supports a maximum string width of 1024. For applications requiring wider strings, you can use several strings to hold the data, use string tables, or use numeric tables, as described in the next section.

Using Numeric Tables as an Alternative to Strings

Since a string is nothing more than a sequence of characters, you can store a string in a numeric table, with each table element holding a character. The advantage of using numeric tables for strings is that a numeric table can store strings of any size. The disadvantages are:

- Memory usage is much greater.

- No string conversion functions are available for numeric tables. An intermediate temporary string would be required to use string commands for these tables.

Strings in ioControl can have a width of up to 1024.

Strings and Multitasking

Although string commands are completed before the current task loses its time slice, it is important to note that a string that is constructed in more than one step may require more than one time slice to complete.

For example, if a string is being constructed in two steps (such as Move String "Hello" and Append String to String " World"), after the first step a task switch could occur, and another chart looking at the resulting string might see "Hello" rather than "Hello World."

If another chart is relying on a completed string, you can use a temporary string for building the string, and then move it to the final string. This idea is illustrated in the following example, where a string variable named MSG_String is built in two steps using a temporary string:

1. Move the string literal "The pressure is " to a temporary variable named sTemp.
2. Append a string variable, sPressure, to sTemp.
3. With the complete string now built, move sTemp to MSG_String.

Adding Control Characters to a String

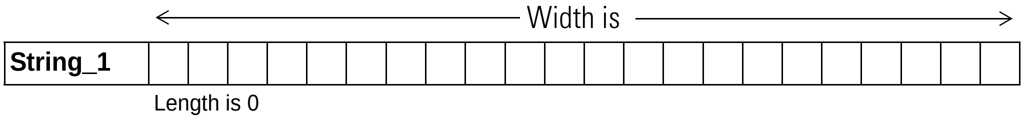
You can input most control characters in a string by typing a backslash (\ ) followed by the two-character hex value of the character. For example, to add an ACK (CTRL+F) character, enter \06 as part of the string.

This technique works for all control characters except null (\00), carriage return (\0D), line feed (\0A), backspace (\08), and CTRL+Z (\1A). To add these characters to a string, you must use the Append Character command.

To input a single backslash in a string, type in a double backslash (\\ ).

Sample String Variable

- Declared Name: String_1
- Declared Width: 22
- Maximum Possible Width: 1024
- Bytes of Memory Required: Declared Width + 4 = 22 + 4 = 26



A string is referred to by its name. Initially the previous string is empty, giving it a length of zero. Later, during program execution, seven characters are added to String_1, increasing its length to seven:

← Width is →

String_1	O	P	T	O		2	2															

← Length is →

Sample String Table

- Declared Name: Promo_Messages
- Declared Width: 26
- Maximum Possible Width: 1024
- Declared Length (Number of indexes, or items, in table): 5
- Maximum Possible Length (Size): 1,000,000
- Bytes of Memory Required: (Declared Width + 4) x Declared Length = (26 + 4) x 5 = 150

← Width is →

Index 0	O	P	T	O		2	2		U	L	T	I	M	A	T	E		I	/	O						
Index 1	I	n	n	o	v	a	t	i	v	e		I	/	O												
Index 2	D	e	l	i	v	e	r	s		c	o	n	t	r	o	l	,									
Index 3		p	r	o	g	r	a	m	m	a	b	i	l	i	t	y	,		a	n	d					
Index 4	e	n	t	e	r	p	r	i	s	e		c	o	n	n	e	c	t	i	v	i	t	y	!		

A string table is a collection of strings. Each string is referred to by the name of the table it is in and the index where it can be found. The length of the table is the number of strings it can hold. Because string table indexes start with zero, indexes can range from zero to the table length minus one.

The width of each string in the table is the same. The length of each string can vary from zero to the configured width of the table.

String Data Extraction Examples

To extract various pieces of information from a string, use the command Find Substring in String. Consider the following example:

	0	1	2	3	4	5	6															
String_1	O	P	T	O		2	2															

One way to get two separate pieces of information from this string is to get characters 0–3 and then get characters 5 and 6, as shown in the following examples.

Find Substring in String: Example 1

	String_1	*string variable*
Start At	0	*integer literal*
Number Of	4	*integer literal*
Move To	Sub_String1	*string variable (width = 5)*

Results in:

	0	1	2	3	4
Sub_String1	O	P	T	O	

Find Substring in String: Example 2

	String_1	*string variable*
Start At	5	*integer literal*
Number Of	2	*integer literal*
Move To	Sub_String2	*string variable (width = 5)*

Results in:

	0	1	2	3	4
Sub_String2	2	2			

String Building Example

Strings are assembled using commands Move String, Append Character to String, and Append String to String. Consider the following original string and the examples that follow:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
String_1	U	L	T	I	M	A	T	E		I	/	O										

Move String

From	OPTO	*string literal*
To	String_1	*string variable*

Results in (note that Move String erased the previous contents of the string):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
String_1	O	P	T	O																		

←Length is→

Append Character to String

From	32	*integer literal (represents a space)*
To	String_1	*string variable*

Results in (note the space character in position 4):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
String_1	O	P	T	O																		

← Length is →

Append String to String

From	22	*string literal*
To	String_1	*string variable*

Results in:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
String_1	O	P	T	O		2	2															

← Length is →

Append Character to String

From	13	*integer literal (carriage return)*
To	String_1	*string variable*

Results in:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
String_1	O	P	T	O		2	2	¶														

← Length is →

Comparison to Visual Basic and C

The following table lists ioControl string commands and their equivalents in Microsoft Visual Basic® and C. If you are using OptoScript, see Appendix F, "OptoScript Language Reference" for additional comparisons.

ioControl Command	Visual Basic	C

Append Character to String	S$ = S$ + Chr$(MyChar%)	i = strlen(str);
		str[i] = MyChar;
		str[i + 1] = 0;

Append String to String	S$ = S$ + “Hello”	strcat(str, “Hello”);
Convert Hex String Number	1% = “&h” + S$	sscanf(str,“%x”,&iNum);
Convert Number to Formatted Hex String	S$ = Hex$(1%)	sprintf(str,“%x”,iNum);

Convert Number to String	S$ = CStr(1%)	sprintf(str,“%d”,iNum);
		sprintf(str,“%f”,fNum);

Convert String to Float	F = CSng(S$)	sscanf(str,“%f”,&fNum);
		fNum = atof(str);

Convert String to Integer 32	I% = CInt(S$)	sscanf(str,“%d”,&iNum);
		iNum = atoi(str);

Get Nth Character	MyByte% = ASC(MID$(Str$,n%,1))	MyByte = str[n];
Get String Length	MyLENGTH% = LEN(Str$)	iLEN = strlen(str);

Get Substring	SubStr$ = MID$(Str$,i,n)	strncpy(subStr,&str[i],n);
		subStr[n] = ‘\0’;

Move String	STR$ = “Hello”	strcpy(strDest,“Hello”);
Test Equal Strings	Equal% = (STR$ = “Hello”)	i = strcmp(str1,“Hello”);
String Equal?	if STR$ = “Hi” then...	if(!strcmp(str1,“Hi”))
String Equal to String Table Element?	if STR$(n%) = “Hi” then...	if(!strcmp(str1[n],“Hi”))

Convert-to-String Commands

The five convert-to-string commands are typically used when printing a number to a port. The ASCII table on the following page shows how various parameters affect the string as it is converted. Note the following:

- Some commands add leading spaces to achieve the specified length. These spaces are indicated with underscores (_).
- Floats (if used) are automatically rounded to integers before conversion except when using the command Convert Number to Formatted Hex String.

	Command Parameters			Convert-to-String Commands				
	Numeric value to be converted	Number of digits right of decimal point	Length	Convert Number to Formatted Hex String (Length 8 required for floats)	Convert Float to String	Convert Number to Hex String	Convert Number to String Field	Convert Number to String
Floats	16.0	1	4	41800000	16.0	10	1.6e+01	1.6e+01
	16.0	2	4	41800000	****	10	1.6e+01	1.6e+01
	-16.0	1	4	C1800000	****	FFFFFFF0	-1.6e+01	-1.6e+01
	1.23	1	4	3F9D70A4	_1.2	1	1.23e+00	1.23e+00
	12.3	1	4	4144CCCD	12.3	C	1.23e+01	1.23e+01
	0.0	1	4	00000000	_0.0	0	0.0e+00	0.0e+00
Integers	16	1	4	0010	16.0	10	_ _16	16
	16	2	4	0010	****	10	_ _16	16
	-16	1	4	FFF0	****	FFFFFFF0	_-16	-16
	0	1	4	0000	0.0	0	_ _ _0	0
	1000	1	2	**	**	3E8	1000	1000

**** Indicates an overflow. The whole-number portion of the resulting string is too long for its space.

ASCII Table

The following table shows ASCII characters with their decimal and hex values. For characters 0–31, equivalent control codes are also listed; for example, a carriage return (character 13) is equivalent to a CTRL+M (shown in the table as ^M).

Dec	Hex	CC	Char
0	00	^@	NUL
1	01	^A	SOH
2	02	^B	STX
3	03	^C	ETX
4	04	^D	EOT
5	05	^E	ENQ
6	06	^F	ACK
7	07	^G	BEL
8	08	^H	BS
9	09	^I	HT
10	0A	^J	LF
11	0B	^K	VT
12	0C	^L	FF
13	0D	^M	CR
14	0E	^N	SO
15	0F	^O	SI
16	10	^P	DLE
17	11	^Q	DC1
18	12	^R	DC2
19	13	^S	DC3
20	14	^T	DC4
21	15	^U	NAK
22	16	^V	SYN
23	17	^W	ETB
24	18	^X	CAN
25	19	^Y	EM
26	1A	^Z	SUB
27	1B	^[	ESC
28	1C	^\	FS
29	1D	^]	GS
30	1E	^^	RS
31	1F	^_	US

Dec	Hex	Char
32	20	Space
33	21	!
34	22	“
35	23	#
36	24	$
37	25	%
38	26	&
39	27	’
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	‘
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

Mathematical Commands

The following commands perform mathematical functions:

Add
Subtract
Multiply
Divide
Modulo
Decrement Variable
Increment Variable
Maximum
Minimum
Round
Truncate
Generate Random Number
Seed Random Number
Absolute Value
Complement
Square Root
Raise to Power
Raise e to Power
Natural Log
Clamp Float Variable
Clamp Float Table Element
Clamp Integer 32 Variable
Clamp Integer 32 Table Element

Trigonometry
Sine
Cosine
Tangent
Arcsine
Arccosine
Arctangent
Hyperbolic Sine
Hyperbolic Cosine
Hyperbolic Tangent

Using Integers

In ioControl, an integer 32 is a 32-bit signed number ranging from -2,147,483,648 to 2,147,483,647 (±2 billion). An integer 64 ranges from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

An integer can only be a whole number (-1, 0, 1, 2, 3, etc.). In other words, integers do not include a decimal point. The result of an integer operation is always an integer, even if it is placed in a float variable. For example, if 9 is divided by 10, the result is zero (0.9 truncated to an integer). To receive a float result, at least one of the operators would have to be a float.

Using Floats

All analog values read from an I/O unit are floating point numbers (floats).

In ioControl, a float is a 32-bit IEEE single-precision number ranging from ±3.402824 x 10^{-38} to ±3.402824 x 10^{38}.

Note that this format guarantees only about six and a half digits of significance in the mantissa. Therefore, mathematical actions involving floats with seven or more significant digits may incur errors after the sixth significant digit. For example, a float-to-integer conversion of 555444333.0 yields 555444416 (note the error in the last three digits).

Controlling Rounding

Use the Round command to control rounding. Note that 1.5 rounds up to 2, 1.49 rounds down to 1. To round down only, divide an integer by an integer (5/3 = 1).

Mixing and Converting Integers and Floats

An analog value read from an I/O unit and put into an integer is converted from float to integer automatically.

To maintain the integrity and accuracy of a numeric type (float or integer), keep all item types the same. For example, use the Move command to copy an integer value to a variable float when you want float calculations.

Logical Commands

The following commands perform logical functions:

Equal?
Equal to Numeric Table Element?
Not Equal?
Not Equal to Numeric Table Element?
Greater?
Greater Than or Equal?
Greater Than Numeric Table Element?
Greater Than or Equal to Numeric Table Element?
Less?
Less Than or Equal?
Less Than Numeric Table Element?
Less Than or Equal to Numeric Table Element?
Within Limits?
Variable False?
Variable True?
Set Variable False
Set Variable True
Get High Bits of Integer 64
Get Low Bits of Integer 64
Make Integer 64
Move 32 bits
Numeric Table Element Bit Clear
Numeric Table Element Bit Set
Numeric Table Element Bit Test

AND
AND?
NOT
NOT?
OR
OR?
XOR
XOR?
Bit Off?
Bit On?
Bit Clear
Bit Set
Bit Rotate
Bit Shift
Bit Test

Bit AND
Bit AND?
Bit NOT
Bit NOT?
Bit OR
Bit OR?
Bit XOR
Bit XOR?
Test Equal
Test Not Equal
Test Greater
Test Greater or Equal
Test Less
Test Less or Equal
Test Within Limits

Understanding Logical Commands

For Condition Blocks, the Instructions dialog box provides options to designate AND or OR for multiple commands. If you have more than one command in the same condition block and you choose the AND option, all of the commands must evaluate true for the block to exit true. If you have more than one command in a condition block and choose the OR option, the block exits true if any of its commands evaluates true.

Logical actions and conditions work with integers, individual bits within an integer, a single digital I/O point, or a digital I/O unit. These values are treated as Boolean; that is, they are either True or False.

For complex logical operations, you may find OptoScript code easier to use than standard ioControl commands. See "Using Logical Operators" on page 11-20 for more information.

Logical True and Logical False

ioControl always returns a value of +1 to indicate True in an integer variable.

A digital input or output that is on also returns a True (+1). Any non-zero value sent to a digital output turns it on. False is defined as zero (0).

For individual bits within an integer variable, bits that are set (1) indicate on. Bits that are cleared (0) indicate off.

While floats can be used in logic, integers are strongly recommended whenever any bits are referenced. Since ioControl does not permit bits in a float value to be altered, float values must be converted to integers before bits can be evaluated. See "Mathematical Commands" on page 10-32 for further information on integers and floats.

Communication Commands

The following commands refer to moving data among entities that store and transfer data:

Communication Open?
Open Outgoing Communication
Listen for Incoming Communication
Accept Incoming Communication
Close Communication
Clear Communication Receive Buffer
Get Communication Handle Value
Set Communication Handle Value
Send Communication Handle Command
Get Available File Space
Get Number of Characters Waiting
Transfer N Characters
Get End-Of-Message Terminator
Set End-Of-Message Terminator

Receive Character
Receive N Characters
Receive String
Receive Numeric Table
Receive String Table
Receive Pointer Table
Transmit Character
Transmit NewLine
Transmit String
Transmit/Receive String
Transmit/Receive Mistic I/O Hex String*
Transmit Numeric Table
Transmit String Table
Transmit Pointer Table

* ioControl Professional only; mistic I/O units only

Communication Handles

A communication handle is a variable in ioControl that stores the parameters needed to connect to a specific entity; the entity may be another device on the network, a file located on the control engine, or some other thing that stores or transfers data. The value of the communication handle variable is a string consisting of the communication parameters separated by colons.

Typically, a communication handle is used to open communication, to transmit and receive data, and to close communication. For example, you might use the TCP communication handle to communicate with another device on the network via TCP/IP, or the FTP communication handle to send data from the brain to a file on a PC.

In many cases you'll define a communication handle variable and never change it, because the same parameters are always needed. However, in some cases it might be useful to change it. For example, if you use the same process to send serial data through two different serial communication modules, you can use Set Communication Handle Value to switch to the other module. Advanced users may also want to use pointers.

Several types of communication handles are available:

TCP	For Ethernet communication For communication with serial modules	See page 10-36. See page 10-38.
File	For creating a file to be stored on the control engine, and reading or writing to a file stored on the control engine	See page 10-42.

FTP	For accessing files on the local file system or another FTP server, including transferring files between the two servers	See page 10-48.
Serial	For using the RS-232 connectors on a SNAP Ethernet-based controller to communicate with serial devices. (Not used for serial communication modules or serial-based I/O units.)	See page 10-50.

Using TCP Communication Handles

The TCP communication handle is used for most Ethernet communication. Any device on the network accessible via TCP/IP can be communicated with using communication commands. For example, data from a serial device can be received through a serial communication module, or communication can be established with a PC on the network.

NOTE: If the two devices that are sharing data are both control engines, it may be easier to use the Scratch Pad areas on each to transfer data, rather than using communication commands. See "I/O Unit—Scratch Pad Commands" on page 10-51 for more information.

Incoming and Outgoing Communication

TCP communication is normally requested by the device that needs the data. For example, an ioControl strategy running on SNAP Ultimate I/O unit A might need the value of a set of variables from another strategy running on Ultimate I/O unit B. Unit A would request communication using the command Open Outgoing Communication. Unit B would Listen for Incoming Communication and then Accept Incoming Communication to establish the connection. Once connected, data could be transmitted and received in both directions.

This scenario is similar to the way we use the telephone. If Joe Wong needs information from Saul Garcia, he requests communication by calling Saul on the phone (opening outgoing communication). If Saul is there (listening for incoming communication), he answers the phone (accepts incoming communication), and the connection is completed. Once connected, both Joe and Saul can talk and listen to transmit and receive information.

Depending on the situation, you may want to have both peers request communication and establish two connections between them. That way if one peer goes offline and then comes online, you can have the flowchart logic set up to immediately open communication again.

In some cases, however, one of the devices may be incapable of requesting communication. A serial device such as a barcode reader attached to a serial communication module on a SNAP Ultimate, SNAP Ethernet, or SNAP Simple I/O unit is an example. The serial device is incapable of requesting Ethernet communication; essentially it acts as a slave device and can only wait until it is asked for data. For serial devices attached to serial communication modules, the control engine needing the serial data must request communication using Open Outgoing Communication. (For serial devices attached directly to a serial port on a SNAP-LCE controller or SNAP Ultimate brain, see "Using Serial Communication Handles to Communicate with Serial Devices" on page 10-50.)

Outgoing Communication For *outgoing* communication (communication you request), the communication handle's value is in the format `tcp:<IP address>:<port>`. Note that `tcp` is all lowercase letters. The following table shows examples of communication handles for outgoing communication.

Outgoing Communication	Communication Handle Value
	Protocol:IP Address:Port
Ethernet/TCP - to another control engine	`tcp:10.192.56.185:22004`
Ethernet/TCP - to a serial communication module on another rack	`tcp:10.192.54.10:22506`

Ethernet/TCP - to a serial communication module on the same rack (uses SNAP Ultimate brain's IP address or loopback IP address)	`tcp:10.192.55.90:22511`
	or
	`tcp:127.0.0.1:22511`

Port Numbers If you are talking to other control engines or other peers on the network, you can use any port number that is not used by another device on the network. For example, do not use the following port numbers, which are used by Opto 22 devices for the purposes shown:

Port #	Purpose
21	FTP (file transfer protocol)
20	FTP (file transfer protocol) for get, send, and dir functions
502	Modbus/TCP hardware and software
2001	Main command processor, OptoMMP-based (also used by SNAP Ethernet I/O and E1/E2 brains)
161	SNMP-based enterprise management systems
22000 22001	Host port (for information on configuring the host port, see form #1440, the *ioManager User's Guide*)

Note that port numbers on the SNAP Ultimate brains, SNAP-LCE controllers, and SNAP PAC controllers are the default ports; some may have been changed for security reasons. (See the section on Security in form #1440, the *ioManager User's Guide*.)

If you do not know what port number to use, ask your network administrator or check the list of standard reserved Ethernet port numbers at http://www.iana.org/assignments/port-numbers to see ports that may apply to your devices on your network.

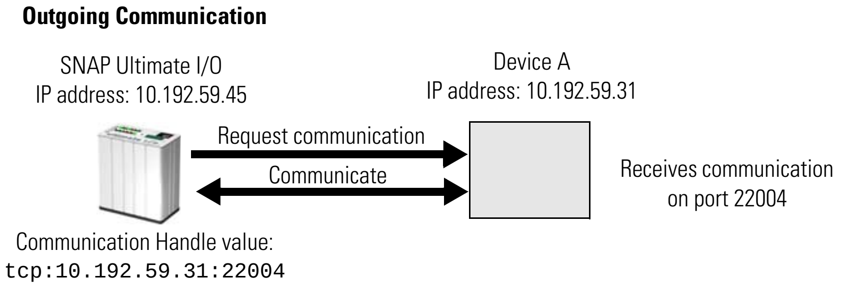
Incoming Communication For *incoming* communication (communication requested by another device), the communication handle value includes just the protocol and the port number: `Protocol:Port`. TCP automatically tracks senders so there is no mixup in the data sent and received. Here are a couple of examples of communication handles for incoming communication:

Incoming Communication	Communication Handle Value
	Protocol:Port
Ethernet/TCP - from another control engine	`tcp:22004`
Ethernet/TCP - from another control engine	`tcp:22005`

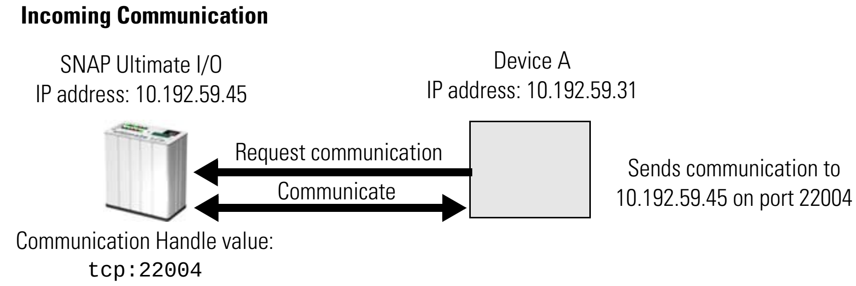
To add a communication handle, see the steps starting on page 9-5.

TCP Communication Handle Examples

The following diagram shows an example of a communication handle value for outgoing communication:



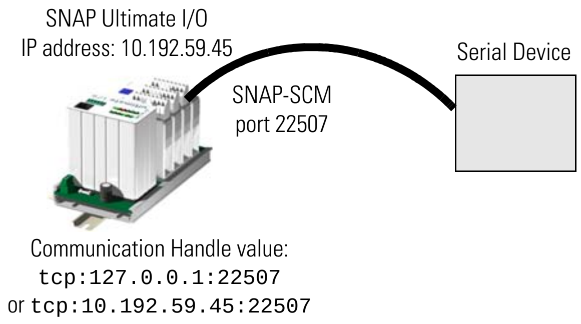
For incoming communication, the communication handle value could be:



Communication Handles for Serial Communication Modules For communication with serial devices through a serial communication module, the communication handle value consists of the IP address of the brain the module is attached to, plus the serial module's port number according to its position on the rack. For port number information, see Opto 22 form #1191, the *SNAP Serial Communication Module User's Guide.*

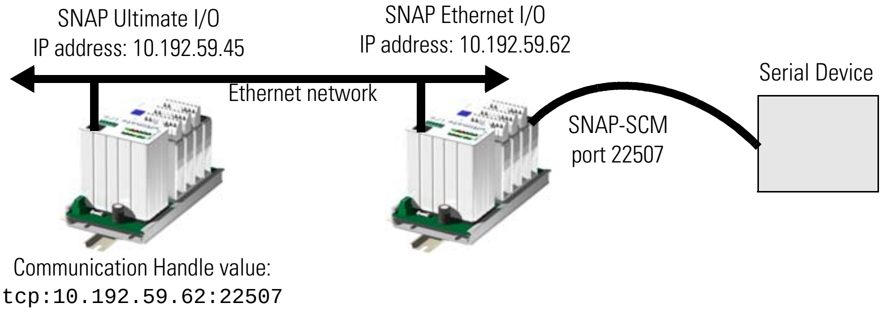
Here are two examples of communication handles for communicating with serial modules, the first showing communication through a module on the same rack as a SNAP Ultimate I/O brain, and the second showing communication through a module on a different rack.

To a Serial Module on the Same Rack



For a serial module on the same rack, use the loopback IP address 127.0.0.1, which tells the brain to talk to its own rack. You can also use the brain's own IP address (in this example, 10.192.59.45), but the loopback address allows you to change the IP address of the brain without having to change the communication handle.

To a Serial Module on a Different Rack

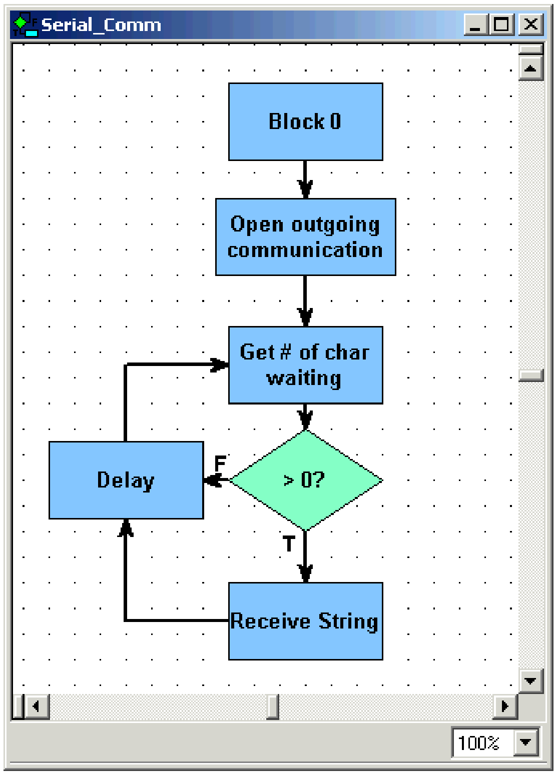


Using Flowcharts to Control TCP/IP Communication

When a control engine is communicating with another device using TCP/IP, it runs an ioControl flowchart or charts to control communication.

- **For outgoing communication,** the flowchart uses the command Open Outgoing Communication to request a connection.
- **For incoming communication,** which is requested by another device, the flowchart must first use Listen for Incoming Communication and then use Accept Incoming Communication to establish a connection.

The control engine's TCP/IP flowcharts should open communication once and then continue to transmit or receive using the communication handle. Constantly opening and closing communication for each transaction wastes time and is inefficient.



As a simple example, the flowchart at right is designed to receive data from a serial device, such as a barcode reader, through a serial communication module on the same rack as a SNAP Ultimate I/O unit.

Communication with serial modules is done via TCP/IP. It can be done from the SNAP Ultimate I/O system itself, as in this example, or from another TCP/IP device on the network.

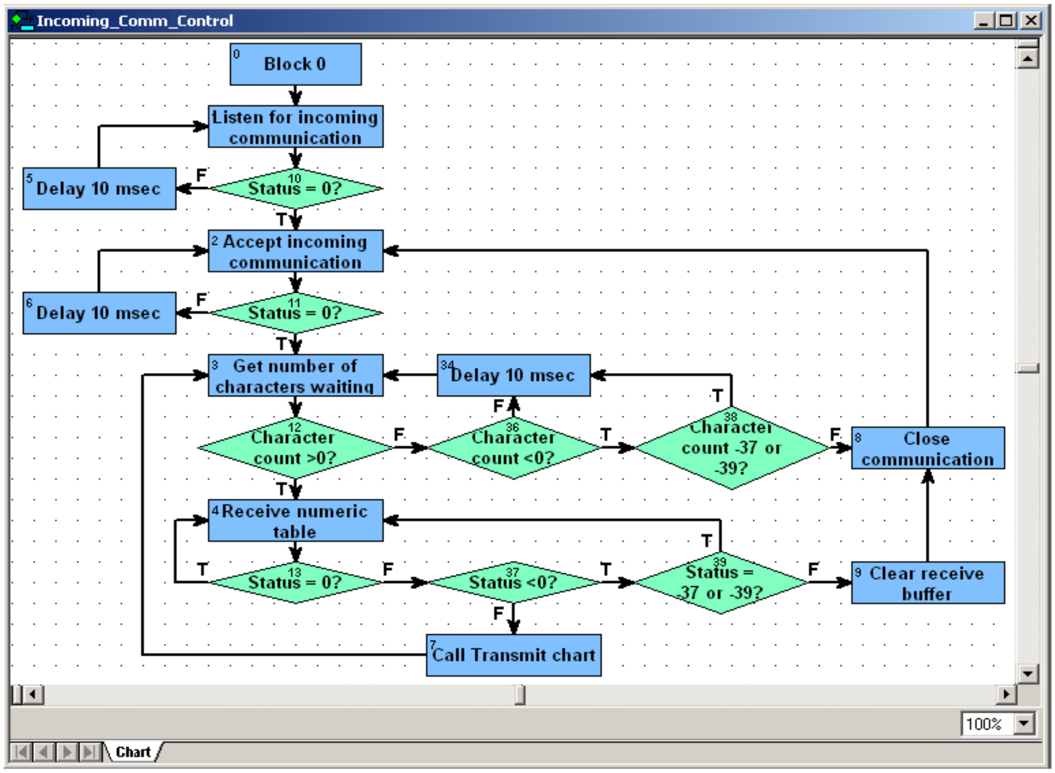
In this example, the Ultimate I/O unit listens for and accepts communication from the serial module using the communication handle:
`tcp:127.0.0.1:22507`
The loopback IP address is used, since the serial module is on the same rack.

When characters are detected in the receive buffer, the I/O unit receives the string and processes it, and then after a short delay checks for another message. Communication remains open for additional messages.

This simple flowchart illustrates the basics of handling communication, without any error checking. However, while receiving and transmitting, TCP/IP control charts should also monitor the status value returned by commands such as Transmit Numeric Table and Receive Numeric Table, and close communication if there are errors. If your ioControl application opens sessions but does not close unused or bad sessions, the maximum number of sessions could be used up. Note that communication should not be closed for timeout errors from Receive commands (error numbers –37 and –39), however, because these errors simply mean that there was no data waiting in the receive buffer for the specified session.

To save time, before using a Receive command (such as Receive String or Receive Numeric Table), TCP/IP charts should use the command Get Number of Characters Waiting. If there are zero characters waiting, then there is no reason to use the Receive command. It is also a good idea to start a down timer, and then loop on the Get Number of Characters Waiting command until either there are more than zero characters waiting, or the timer expires.

The following chart is also an example of TCP/IP communication, but with three differences: in this case another device is requesting communication, the peer will both transmit and receive, and error checking is shown for the commands Get Number of Characters Waiting and Receive Numeric Table. Similar error checking should be used for transmit commands, which in this strategy are in another chart.



If the devices communicating with each other are all SNAP Ultimate I/O systems, be careful not to have one of them send information to another faster than the receiving system can pull it out of its receive buffer. When a control engine receives data from the Ethernet network, it holds the data in memory (the buffer) until the ioControl flowchart removes it with a Receive command. If the data is not removed fast enough, the receive buffer will fill up. You can prevent this problem by transmitting the information less frequently. If the same chart is transmitting and receiving, you can alter the chart so that it receives more often than it transmits.

Ethernet Connections and Ports

The number of Ethernet connections available varies by hardware, type, and in some cases firmware version, such as 64 on current SNAP Ultimate I/O systems.

The host connection is on port 22000 or 22001 and is used to communicate with the PC that runs ioControl. These port numbers are reserved for this purpose, but they can be configured. For information see Opto 22 form #1440, the *ioManager User's Guide.*

Peer-to-peer connections can be on any other port number that is not already used on the network; these connections are used to communicate with other control engines or other devices on the network. When you assign port numbers for peer-to-peer connections, make sure you do not assign port numbers that devices on your network may use for specific purposes. For a list of standard Ethernet port numbers, refer to: http://www.iana.org/assignments/port-numbers

IMPORTANT: If you are using TCP/IP connections, consult commercially available texts on TCP/IP that discuss client/server architectures, so you'll understand how the protocol works and what to expect during communication.

Using the Control Engine's File System

The memory in SNAP PAC and SNAP-LCE controllers and in SNAP Ultimate controller/brains includes a substantial area (about 4 MB on a SNAP-PAC-S1, 2 MB on a SNAP PAC R-series controller or SNAP Ultimate brain, and 1 MB on a SNAP-LCE) available for file storage. Any types of files can be stored there, and files can be sorted into directories or folders just as they can on a PC. These stored files are then available for use; for example, the control engine can read them, add data to them, and even send data from them via FTP to another device on the network.

Note: Certain FTP commands may also be useful when dealing with files, even if the files are all local. For example, the dir command is available with comm handles.

The file communication handle is used to create, write to, and read from stored files on the control engine. The format for the handle's value is: `file:<open mode>,<filename>`

Note that `file` is all lowercase. Open modes are:

Open mode	Description
r	Opens a file for reading. If the file does not exist or cannot be found, the open call fails.
w	Creates a new file and opens it for writing. If the file already exists, its contents are destroyed.
a	Opens a file for writing at the end of the file (appending). If the file doesn't exist, it is created.

Here are some examples of file communication handle values:

Creates the file myfile.txt and opens it for writing.	`file:w,myfile.txt`
Opens the existing file myfile.txt so data can be appended to it.	`file:a,myfile.txt`
Creates the file Temperature data. txt in the directory Data_files and opens it for writing. If the directory doesn't exist, it is created.	`file:w,/Data_files/Temperature data.txt`
Opens the file Temperature data.txt in the directory Data_files for reading.	`file:r,/Data_files/Temperature data.txt`

Keep the following limitations in mind as you use the file communication handle:

Maximum length for filenames and directory names	127 characters
Filename characters allowed	All ASCII characters except *, ?, null, and /
Path name component separator	/
Maximum number of files and directories that can be open simultaneously	16
Maximum directory depth	Limited only by available memory
Maximum number of files	Limited only by available memory. Each file uses 516 bytes of overhead plus its number of bytes rounded up to the nearest multiple of 516 bytes.
Maximum number of directories	Limited only by available memory. Each directory uses 516 bytes.
Maximum amount of memory available in the control engine's file system	Approximately 4 MB on a SNAP-PAC-S1, 2 MB on a SNAP PAC R-series controller or SNAP Ultimate brain, or 1 MB on a SNAP-LCE controller (varies slightly depending on the control engine firmware version)

If power to the control engine is turned off, files are destroyed unless they have been saved to flash memory. See "Commands Relating to Permanent Storage" on page 10-17 for information on saving files to flash using ioControl commands.

Working with Files in Your Strategy

The commands you use with a communication handle vary according to the action (the open mode) defined in the handle's value. To change actions—from read to write, for example—you can use Set Communication Handle Value to change the handle's value.

Commands to use with any file communication handle	Commands to use with file comm handles in Read mode	Commands to use with file comm handles in Write or Append mode
Open Outgoing Communication Communication Open? Send Communication Handle Command (*delete*, *getpos*, or *setpos*) Set Communication Handle Value Set End-of-Message Terminator Get Available File Space Get Number of Characters Waiting Close Communication	Receive Numeric Table Receive Pointer Table Receive String Receive String Table Send Communication Handle Command (*find*)	Transmit Character Transfer N Characters Transmit NewLine Transmit Numeric Table Transmit Pointer Table Transmit String Transmit String Table

To work with files in your strategy, first use the command Open Outgoing Communication.

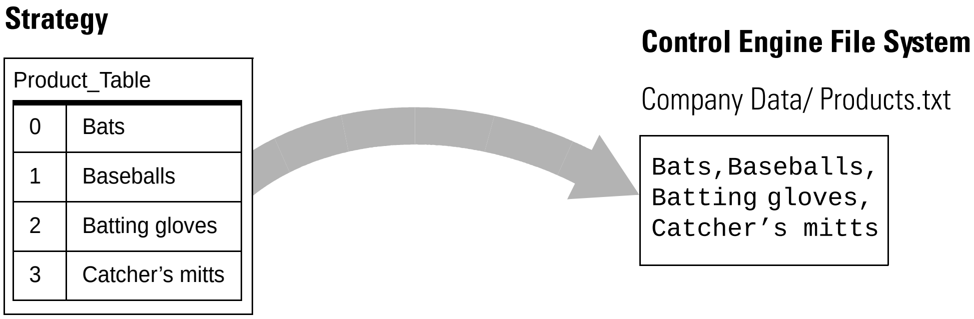
Writing to a File For example, suppose you have data in your strategy in a string table (Product_Table) that you want to write to a file (Products.txt) in a directory (Company Data) on the control engine. Here's how you would do it:

1. Use Open Outgoing Communication to open a file communication handle. The value of the handle would be: `file:w,/Company Data/Products.txt`

2. Use the condition Communication Open? to make sure the communication handle opened.

3. To put the data from the string table into a comma-delimited file (which is easy to open in database software), use the command Set End-Of-Message Terminator to indicate that a comma should be used as the delimiting character.

4. Use the command Transmit String Table to transmit data from Product_Table directly into the Products.txt file. Items from the string table are separated by commas in the file.



Product_Table

0	Bats
1	Baseballs
2	Batting gloves
3	Catcher’s mitts

```
Bats,Baseballs,
Batting gloves,
Catcher’s mitts
```

5. Use the command Close Communication to close the communication handle.

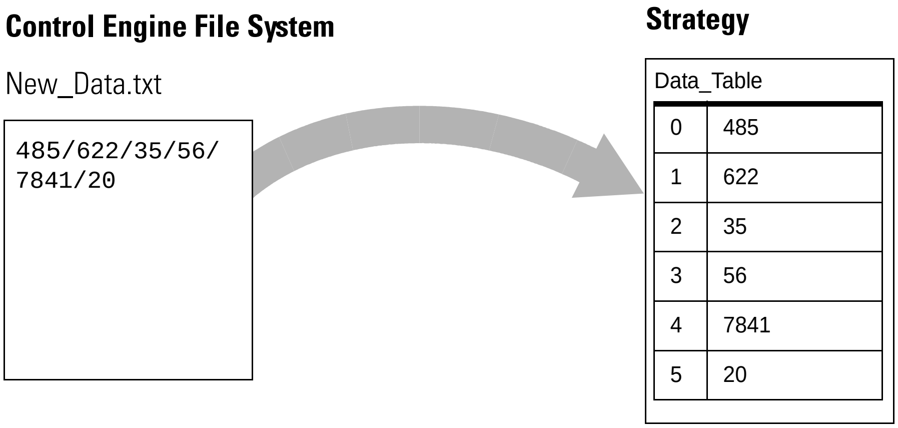
Reading a File As another example, suppose you have a file on the control engine that was placed there via FTP. (See the next section for details on using the FTP communication handle.) You want to read this file (New_Data.txt) and place the data in it into a string table (Data_Table) in your strategy.

1. Use Open Outgoing Communication to open a file communication handle. The value of the handle would be: `file:r,New_Data.txt`

2. Use the condition Communication Open? to make sure the handle opened.

3. Use the command Set End-Of-Message Terminator to indicate what character in the New_Data.txt file should be read as the delimiting character. (In the example shown below, it’s a slash.)

4. Use the command Receive String Table to receive the data from New_Data.txt directly into Data_Table.



```
485/622/35/56/
7841/20
```

0	485
1	622
2	35
3	56
4	7841
5	20

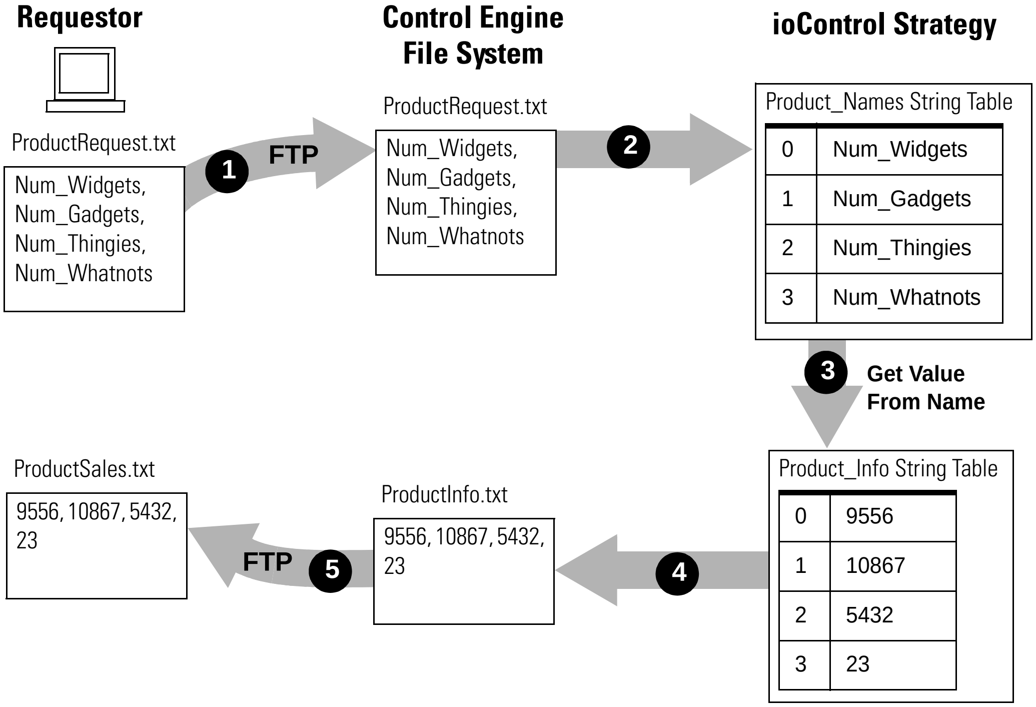
Notice the numbers used in this example. These are numbers represented as strings. For the purpose of storing and sending data, this is the simplest way to represent them. If you need to use them in calculations, however, you must first convert them to numeric values. You can do so in your ioControl strategy by using a command such as Convert String to Float or Convert String to Integer 32. (See "String Commands" on page 10-23 for more information.)

5. Finally, close the communication handle by using the command Close Communication.

A More Complex Example. Here's a more complex example which shows the actual OptoScript code. In this example, someone on the network needs the value of several variables in the ioControl strategy running on the control engine. This person has sent to the control engine via FTP a comma-delimited text file (ProductRequest.txt) containing the names of the variables. (See page 10-48 for information on FTP.)

This section of the ioControl strategy reads the variable names from the text file in the control engine's file system and places the names in a string table (Product_Names). Next, the strategy uses the command Get Value From Name to place the *values* of the variables into another table (Product_Info). The data from this table is then written to another text file (ProductInfo.txt) on the control engine, which can later be sent via FTP to the person who requested the data.

The whole operation (including the FTP portions, which are not covered in this code example), might look like this:



Product_Names String Table

0	Num_Widgets
1	Num_Gadgets
2	Num_Thingies
3	Num_Whatnots

Product_Info String Table

0	9556
1	10867
2	5432
3	23

The OptoScript code in this example also makes use of the command Set Communication Handle Value to change the value of a specific communication handle during the operation.

Sets the value for the communication handle chAFile.	`SetCommunicationHandleValue("file:r,ProductRequest.txt",chAFile);`

Opens the communication handle and checks to make sure it opened.	`status = OpenOutgoingCommunication(chAFile);`
	`if (status == 0) then`

Sets the end-of-message terminator to a comma because the file to be read is comma-delimited.	` SetEndOfMessageTerminator(chAFile, ',');`
Reads the contents of the file Product_Names into a string table.	` status = ReceiveStrTable(4,0,Product_Names,chAFile);`

Loops through the items in Product_Names table and places the values they represent into another string table, Product_Info. (Note that the "numbers" in the Product_Info table are not true numbers, but string representations of numbers.)	` index = 0;`
	`  while ((index < 4) and (status == 0))`
	`    status = GetValueFromName(Product_Names[index], Product_Info[index]);`
	`    index = index + 1;`
	`  wend`

Closes communication.	` status = CloseCommunication(chAFile);`
Changes the value of the communication handle; it is now set to write to the file ProductInfo.txt on the control engine.	` SetCommunicationHandleValue("file:w,ProductInfo.txt",chAFile);`

Opens the communication handle and checks to make sure it opened.	` status = OpenOutgoingCommunication(chAFile);`
	` if (status == 0) then`

Makes the ProductInfo.txt file comma-delimited, too.	`  SetEndOfMessageTerminator(chAFile, ',');`

Writes the data from the Product_Info string table into the ProductInfo.txt file on the control engine.	`  status = TransmitStrTable(4,0,Product_Info,chAFile);`
	` endif`
	`endif`

Deleting Files and Moving Within Them Another command you'll find useful with file communication handles is Send Communication Handle Command. Using these commands, you can delete files, find a position within the file, and jump to a specific position within the file. See the *ioControl Command Reference* or online Help for details.

Moving Files via FTP

As explained in the previous section ("Using the Control Engine's File System," starting on page 10-42) the control engine's memory includes a substantial area available for file storage. You can move files to and from file storage using the File Transfer Protocol (FTP):

- **To move files to and from file storage using another device such as a PC,** use any standard FTP software. See instructions in Opto 22 form #1440, the *ioManager User's Guide*. A maximum of five devices can FTP files to a control engine simultaneously.
- **To move data to and from file storage programmatically,** a strategy can request or send a file using an FTP communication handle, explained in this section. A maximum of 16 communication handles can be used simultaneously to move data via FTP.

FTP also allows you to get a directory listing of files on a remote server or in the local file storage area.

FTP Communication Handle Examples

The value for the FTP communication handle is in the format:

```
ftp:<IP address>:<port>,<username>,<password>,<optional timeout>
```

`ftp`	Use all lowercase letters.
`IP address`	IP address of the destination device (where the file will go).
`port`	Default port is 21, with 20 for the data port.
`username` `password`	Enter the username and password set up for the destination device. If the destination device is another control engine or the local server, use anything except an empty string.
`optional timeout`	Specify a communication timeout value in seconds, or leave this parameter out to use the default of 30 seconds.

Here are two examples of FTP communication handle values:

`ftp:10.22.55.35:21,jsmith,2towers,60` (timeout increased to 60 seconds)

`ftp:10.192.54.195:21,m,m` (no timeout specified)

Using FTP Communication Handles in Your Strategy

Suppose you have data you need to send via FTP to a device on the network whose IP address is 10.192.56.45. Your username on this device is JoeG and your password is hello. You expect the default timeout of 30 seconds to be adequate.

There are two ways you can send the data: all at once, or in pieces over time. If your data file is less than 1 MB in size, you can send it all at once. If it is larger, or if you want to append additional data to a file that already exists, you send it in pieces.

Sending the Data All At Once This method is better for small data files; files larger than 1MB may take a long time to transfer, and the control engine may become unresponsive during the process.

Suppose you want to place the data into a file (HunkoData.txt) on the device; the data is currently in the file UseThisData.txt on the control engine. Here's how you would send the data in your ioControl strategy:

1. Use the command Open Outgoing Communication to open an FTP communication handle. The value of the handle would be: `ftp:10.192.56.45:21,JoeG,hello`
2. Use the condition Communication Open? to make sure the communication handle opened.
3. Use the command Send Communication Handle Command to specify the filename and send the file in one step. The communication handle you send is:

   ```
   send:UseThisData.txt,HunkoData.txt
   ```

   If the remote filename already exists, it is overwritten.
4. When you have finished sending data from the file, use the command Close Communication to close the communication handle.

Sending the Data in Pieces To append data to an existing file, or to send a very large file (larger than 1 MB), you can send data in pieces.

Suppose you want to append data from a strategy variable or table to an existing file named HunkoData.txt on the device. Here's how you would send the data:

1. Use the command Open Outgoing Communication to open an FTP communication handle. The value of the handle would be: `ftp:10.192.56.45:21,JoeG,hello`
2. Use the condition Communication Open? to make sure the communication handle opened.
3. To specify the filename on the remote server, use the command Send Communication Handle Command. The communication handle command you send is:

   ```
   dest:HunkoData.txt
   ```
4. Use a Transmit command (such as Transmit String or Transmit Numeric Table) to send the data. In the command, use the name of the FTP communication handle you just opened.

   The data is appended to the file. (If the remote filename is new, the file is created and the data placed in it.)
5. When you have finished sending data, use the command Close Communication to close the communication handle.

If you are sending a large file from the control engine to the device, you would need to open up two communication handles: an FTP handle just like the one in the example above, and a File handle for the file on the control engine. Then use the Transfer N Characters command to send the file in chunks.

For another example, see the diagram in "A More Complex Example." on page 10-45, which shows how FTP communication handles and file communication handles might be used together. See the *ioControl Command Reference* or online Help for detailed information on commands.

Retrieving a Directory Listing

The OptoScript code in this example makes use of the command Set Communication Handle Value and the dir option to retrieve a directory listing.

Sets the value for the communication handle chAFile.	`// Configure the chFTP comm handle to log into itself. The username and`
	`// password don't matter--they just can't be left empty. We'll use the`
	`// loopback address of 127.0.0.1 so this code is more portable.`
	`SetCommunicationHandleValue("ftp:127.0.0.1:21,noimporta,whocares", chFTP );`

Opens the communication handle and checks to make sure it opened.	`// Open the communication handle (log in to the local server)`
	`nStatus = OpenOutgoingCommunication( chFTP );`
	`if (nStatus == 0) then`

Requests directory listing, returns number of listings.	`  nStatus = SendCommunicationHandleCommand( chFTP, "dir" );`

Makes sure "dir" worked—firmware must be version 7.1 or greater.	`  if (nStatus >= 0) then`
	`    nFileCount = nStatus;`

Reads in listings to stList.
 	
Note: Each listing from an Opto 22 device server ends with these three hex bytes: 0D 0A 00
	`    // Set the EOM character to the last byte of the listing: 00`
	`    SetEndOfMessageTerminator( chFTP, 0x00 );`
	`    nStatus = ReceiveStrTable( nStatus, 0, stList, chFTP );`
	
	`    if ( nStatus == 0 ) then`

Optional: parses each listing into separated tables for:
1. date/time stamp (first 17 characters)
2. file size (next 23 characters)
3. filename (remaining characters) and removes the 0D 0A left over on the end of the filename.
	`      for nIndex = 0 to (nFileCount - 1) step 1`
	`        GetSubstring( stList[nIndex], 0, 17, stDateTimeStamps[nIndex] );`
	`        GetSubstring( stList[nIndex], 18, 23, sTemp );`
	`        ntSizes[nIndex] = StringToInt32( sTemp );`
	`        GetSubstring( stList[nIndex], 39, 1000, stFilenames[nIndex] );`
	
	`        nLength = GetStringLength( stFilenames[nIndex] );`
	`        GetSubstring( stFilenames[nIndex], 0, nLength - 2,`
	`                      stFilenames[nIndex] );`
	`      next`

Further parsing could be done on the date/time stamps, depending on how those values will be used.	`    endif`
	`  endif`
	`endif`
	`CloseCommunication( chFTP );`

Using Serial Communication Handles to Communicate with Serial Devices

RS-232 serial connectors are located on the top of SNAP PAC controllers, SNAP-LCE controllers, and SNAP Ultimate controller brains. These ports can be used for maintenance, such as loading new firmware, or for Point-to-Point Protocol (PPP) communication via modem, but they can also be used to send or receive data directly from a serial device, such as barcode readers, weigh scales, or any intelligent device with a serial port. Serial communication handles are used to communicate with these devices.

You must configure the ports specifically for this use. See the *ioManager User's Guide* for instructions. On a SNAP-LCE or SNAP Ultimate brain, firmware version 5.1c or newer is required.

IMPORTANT: *Serial communication handles are used only for direct connection to serial devices. If you are connecting to serial devices through serial communication modules on the I/O unit, use a TCP communication handle instead. See page 10-36.*

NOTE: Serial I/O units are wired to the RS-485 port on the controller and do not require serial communication handles. Communication is defined when the I/O unit is configured.

Serial Communication Handle Examples

The value for the serial communication handle is in the format:

```
ser:<port number>,<baud rate>,<parity>,<data bits>,<stop bits>
```

`port number`	0 or 1.
`baud rate`	115200, 76800, 57600, 38400, 19200, 9600, 4800, 2400, 1200, or 300
`parity`	n, o, or e (none, odd, or even)
`data bits`	8 or 7
`stop bits`	1 or 2

Here is an example of a serial communication handle value:

`ser:0,115200,n,8,1` (port 0, baud rate of 115,200, no parity, 8 data bits, and 1 stop bit)

Using Serial Communication Handles in Your Strategy

To use a serial communication handle in your strategy, first use the command Open Outgoing Communication. Verify that the communication handle opened by using the condition Communication Open? Then use Transmit, Transfer, and Receive commands to send or receive data as necessary. Remember to check for any errors. When you have finished sending and receiving data, use the command Close Communication to close the communication handle.

Make sure your strategy receives data promptly. Incoming serial communication is buffered up to 127 characters. If more than 127 characters come in before the strategy receives them, the additional characters are lost.

See the *ioControl Command Reference* or online Help for details on using specific commands.

I/O Unit—Scratch Pad Commands

The following commands are used for peer-to-peer communication for sharing strategy data with other Opto 22 memory-mapped controllers on the network. These commands are used by ioControl to read or write to the Scratch Pad area in the memory map of a SNAP Ultimate I/O unit or a SNAP PAC or SNAP-LCE controller.

Since each read (get) or write (set) command is completed before another occurs, commands cannot interfere with each other. For example, a get command won't read a partial string while a set command is writing the string.

Set I/O Unit Scratch Pad Bits from MOMO Mask
Get I/O Unit Scratch Pad Bits
Set I/O Unit Scratch Pad Integer 32 Element
Get I/O Unit Scratch Pad Integer 32 Element
Set I/O Unit Scratch Pad Integer 32 Table
Get I/O Unit Scratch Pad Integer 32 Table
Set I/O Unit Scratch Pad Float Element
Get I/O Unit Scratch Pad Float Element
Set I/O Unit Scratch Pad Float Table
Get I/O Unit Scratch Pad Float Table
Set I/O Unit Scratch Pad String Element
Get I/O Unit Scratch Pad String Element
Set I/O Unit Scratch Pad String Table
Get I/O Unit Scratch Pad String Table

Since these are I/O unit commands, when they are used for a SNAP PAC or SNAP-LCE controller, you must have already configured an I/O unit to represent the controller. Configure the controller as a SNAP-UP1-M64 with the IP address of the control engine.

Also because these are I/O unit commands, remember to check all return values and errors to make sure the command was successful. If a command variable contains a value that is obviously wrong—for example, a memory map address in an incorrect format—communication to the I/O unit will be automatically disabled.

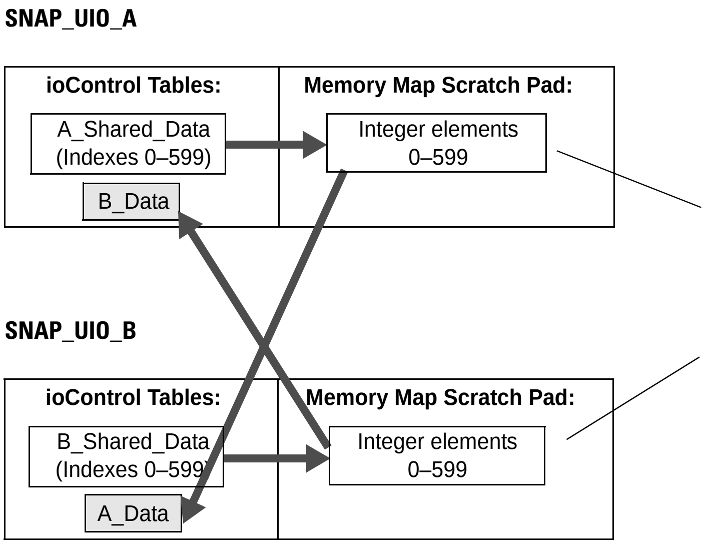
Each controller or SNAP Ultimate I/O unit running an ioControl strategy can place data in its own or another's Scratch Pad area, and each can retrieve data that has been placed in the Scratch Pad area by other devices using other applications. Using these commands eliminates the need to open communication handles (see "Communication Commands" on page 10-35), thus speeding up peer-to-peer communication.

The memory map Scratch Pad area supports four data types: bits, integer 32s, floats, and strings.

- For details on the Scratch Pad area, see Opto 22 form 1440, the *ioManager User's Guide*.
- For the complete memory map, see form 1465, the *OptoMMP Protocol Guide.*
- For details on using the Scratch Pad for peer-to-peer communication with a controller, see the controller's user's guide.

The following page shows a simple example of how Scratch Pad area data exchange would work between two SNAP Ultimate I/O systems.

Create two tables for SNAP_UIO_A, one for its own data that will be shared (A_Shared_Data) and another for data it will read from SNAP_UIO_B (B_Data). Also create two tables for SNAP_UIO_B, one for its own data (B_Shared_Data) and one for SNAP_UIO_A's data (A_Data).



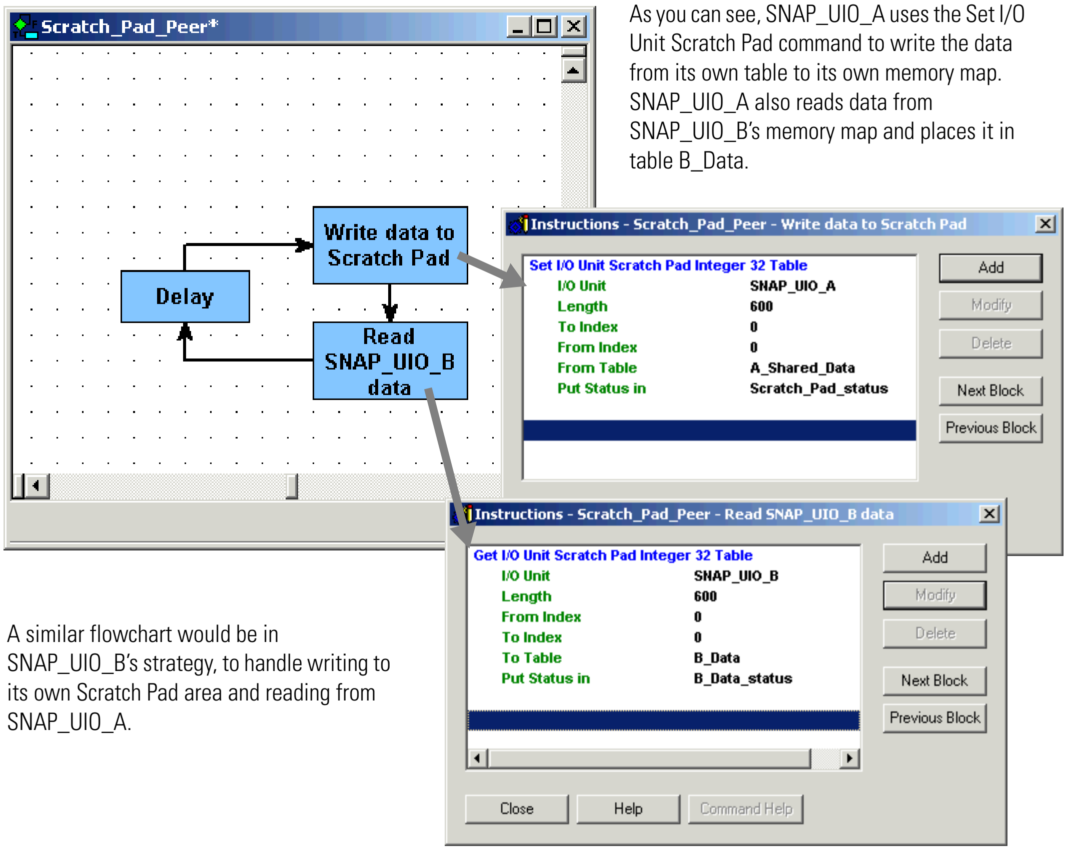
Suppose SNAP_UIO_A and SNAP_UIO_B are sharing 600 integer elements of the 3072 integer elements available in the Scratch Pad.

SNAP_UIO_A writes data from its own ioControl strategy table to its own Scratch Pad area, which SNAP_UIO_B can then read.

Meanwhile, SNAP_UIO_B writes data from its B_Shared_Data table to its Scratch Pad area, which SNAP_UIO_A reads.

NOTE: When the SNAP Ultimate I/O unit is writing to its own Scratch Pad, use the loopback IP address, 127.0.0.1

This portion of the flowchart in SNAP_UIO_A (without error checking) might look like this:


Scratch_Pad_Peer*
Write data to Scratch Pad
Delay
Read SNAP_UIO_B data
Instructions - Scratch_Pad_Peer - Write data to Scratch Pad
Add
Modify
Delete
Next Block
Previous Block
Instructions - Scratch_Pad_Peer - Read SNAP_UIO_B data
Close
Help
Command Help

Set I/O Unit Scratch Pad Integer 32 Table

I/O Unit	SNAP_UIO_A
Length	600
To Index	0
From Index	0
From Table	A_Shared_Data
Put Status in	Scratch_Pad_status

Get I/O Unit Scratch Pad Integer 32 Table

I/O Unit	SNAP_UIO_B
Length	600
From Index	0
To Index	0
To Table	B_Data
Put Status in	B_Data_status

As you can see, SNAP_UIO_A uses the Set I/O Unit Scratch Pad command to write the data from its own table to its own memory map. SNAP_UIO_A also reads data from SNAP_UIO_B's memory map and places it in table B_Data.

A similar flowchart would be in SNAP_UIO_B's strategy, to handle writing to its own Scratch Pad area and reading from SNAP_UIO_A.

I/O Unit—Event Message Commands

The following commands refer to event messages sent from SNAP PAC R-series, SNAP Ultimate or SNAP Ethernet I/O units: They do not apply to mistic event/reactions.

Get I/O Unit Event Message State	Set I/O Unit Event Message State
Get I/O Unit Event Message Text	Set I/O Unit Event Message Text

A SNAP Ultimate or Ethernet I/O system can send a message as a response to an event that occurs within strategy logic. For example, if pressure in a pipe reaches a certain level, the system can send a warning email message to a technician. Or data about a process can be streamed to a computer every 30 seconds for monitoring.

CAUTION: *Events and reactions, including event messages, that are processed separately from your strategy can conflict with strategy logic. If you are using ioControl, use strategy logic instead of local events and reactions on the I/O unit, or be very careful that local events and reactions do not conflict.*

Use ioManager to configure event messages, following the steps in Opto 22 form 1440, the *ioManager User's Guide*. You can configure the following types of event messages:

- Email or paging messages sent to a person
- Simple Network Management Protocol (SNMP) traps sent to an enterprise management system
- Serial messages sent through a serial communication module to a serial device
- Streamed data sent to a device for processing by a software application.

I/O Unit—Event Message commands are commonly used to find out the state or text of an event message, or to set its state. Event messages can be in the following states:

- **Active**—The message has been triggered by the event. If it was configured to be sent just once, it has been sent. If it was configured to be sent at intervals, it is continuing to be sent.
- **Inactive**—The message is not currently triggered by the event.
- **Acknowledged**—The message has been triggered by the event but has been acknowledged by the receiver so it will not be sent again. (Acknowledgments occur only if the receiving application writes them to the brain's memory map.) Acknowledged is functionally equivalent to Inactive, but can be useful in some cases to determine whether the receiver has received the message.

The command Set I/O Unit Event Message Text is used to dynamically change a message or to "recycle" a message if you run out of event messages on an I/O unit (128 event messages are available for each I/O unit). For more information, see individual commands in the *ioControl Command Reference* or online help.

I/O Unit—Memory Map Commands

The following commands refer to the memory map in an Opto 22 memory-mapped device, either a controller or an I/O unit. In the case of a controller/brain, they can refer to the device's own memory map or a memory map on another device.

Read Number from I/O Memory Map
Read Numeric Table from I/O Memory Map
Read String from I/O Memory Map
Read String Table from I/O Memory Map
Write Number to I/O Memory Map
Write Numeric Table to I/O Memory Map
Write String to I/O Memory Map
Write String Table to I/O Memory Map

Memory map commands make it possible for advanced users to read from or write to any Opto 22 memory-mapped device, such as a SNAP PAC controller or a SNAP Ultimate or Ethernet I/O unit. You can use these commands to read or write to any address within the memory map. The commands are especially useful for reading data from a SNAP device using newer features that may be available in the memory map but are not yet incorporated into ioControl.

NOTE: If you are reading or writing to the device's Scratch Pad area, use the I/O Unit—Scratch Pad commands instead (see page 10-51). If you are changing event messages, use the I/O Unit—Event Message commands instead (page 10-54).

Before you use these commands with a SNAP PAC or SNAP-LCE controller, you must have already configured an I/O unit to represent the controller. Configure it as a SNAP-UP1-M64 with the controller's IP address.

When you use these commands, make sure that you read or write the correct type of data (integer, float, string) to match the specified memory map address. The control engine doesn't know what type of data is in any particular address, so it cannot convert the data type.

Since these are I/O unit commands, remember to check all return values and errors to make sure the command was successful. If a command variable contains a value that is obviously wrong—for example, a memory map address in an incorrect format—communication to the I/O unit will be automatically disabled.

See the *OptoMMP Protocol Guide* (Opto 22 form 1465) to determine the memory map addresses and data types you need to use.

Error Handling Commands

The following commands refer to handling errors:

Error?
Error on I/O Unit?
Copy Current Error to String
Remove Current Error and Point to Next Error
Clear All Errors
Caused a Chart Error?
Caused an I/O Unit Error?
Add User Error to Queue
Add User I/O Unit Error to Queue
Add Message to Queue

Get Error Count
Get Error Code of Current Error
Get Severity of Current Error
Get ID of Block Causing Current Error
Get Line Causing Current Error
Get Name of Chart Causing Current Error
Get Name of I/O Unit Causing Current Error
Disable I/O Unit Causing Current Error
Enable I/O Unit Causing Current Error
Stop Chart on Error
Suspend Chart on Error

All good programmers must deal with errors. These error handling commands are used to monitor errors, figure out which I/O unit caused an error, disable or re-enable the unit, and clear errors and other messages from the message queue. For a simple example of an error handler chart, see page 4-13. For more on the message queue, see "Queue Messages" on page B-2.

You can use the command Add User Error to Queue to add your own information, warning, or error message to the queue. This command can be helpful in troubleshooting.

Pointer Commands

The following commands are used with pointers:

Move to Pointer
Move to Pointer Table
Move from Pointer Table Element
Get Pointer From Name

Clear Pointer
Clear Pointer Table Element
Pointer Equal to NULL?
Pointer Table Element Equal to NULL?

See also:

"Understanding Pointers"

"Advantages of Using Pointers"

"Referencing Objects with Pointers"

Understanding Pointers

Like integer and float variables, a pointer variable stores a specific number. However, the number is not data—it is the memory location (address) of data. The pointer “points” to data rather than containing the data.

A pointer in ioControl can point to many different types of objects:

- Another variable
- A Digital Point or object
- An Analog Point or object
- An I/O unit
- A chart.

Pointers cannot point to other pointers, however. If you try to move a pointer to a pointer, ioControl just duplicates the existing pointer.

The following table lists the objects that pointers can point to:

Digital Objects	Analog Objects	I/O Units	Variables	Other Objects
Digital Input Digital Output Counter Quadrature Counter	Analog Input Analog Output	SNAP-ENET-D64 SNAP-UP1-D64 SNAP-B3000-ENET, SNAP-ENET-RTC SNAP-UP1-ADS SNAP-UP1-M64 SNAP-ENET-S64	Integer Variable Float Variable String Variable Pointer Variable Down Timer Variable Up Timer Variable Integer Table Float Table String Table Communication Handle	Chart

Advantages of Using Pointers

For certain types of operations, pointers can speed up programming and make the strategy more efficient. Pointers are usually recommended only for experienced programmers, however, because their misuse can result in unpredictable behavior. They also complicate strategy debugging. If you use too many pointers, it’s easy to lose track of what’s pointing to what.

If you choose to use pointers, be sure you use the text tool to document your charts in detail.

Referencing Objects with Pointers

There are two types of pointers—pointer variables and pointer tables.

Pointer Variables A pointer variable contains a single pointer to a single object. You can set the initial value for a pointer variable when you configure it, or you can set it later by using the command Move to Pointer.

Once the initial value is set, you can reference it using any command you would use for that type of object. For example, if the pointer points to a string variable, you can use any command for the pointer that you would normally use for a string variable, such as Append String to String or Convert String to Float.

Pointer Tables A pointer table contains a list of objects of different types, each of which can be pointed to. For example, the object at index 0 could be a chart, the object at index 1 a Digital Point, and the object at index 2 a string variable. An example of using a pointer table for indexing is shown on page 4-19.

When you create a pointer table, no initial values are set. You can use the Move to Pointer Table command to set individual values in the table.

A pointer table element cannot be directly referenced. It must be copied to a pointer variable first, using the command Move From Pointer Table Element. Once it is in the pointer variable, you can reference it as you would any object of that type. For example, if index 3 in a pointer table points to a counter input, use Move From Pointer Table Element to put the counter input address in a pointer variable. Then you can use any command for the pointer variable that you would normally use with a counter input, such as Start Counter or Clear Counter.

PID—Ethernet Commands

The following commands are used with PID loops on SNAP Ethernet-based I/O units. For mistic I/O units, see "PID—Mistic Commands" on page 10-62. Since PID loops are configured and tuned in Configure Mode while creating your ioControl strategy, you may not need to use these commands. PID commands are typically used to change input or output location (for example, if the input is on another I/O unit) or to change tuning parameters based on a change that occurs while the strategy is running (such as a recipe change).

For steps to configure and tune PIDs, see page 6-29. For more information about PID loops on Ethernet-based I/O units, how to tune them, and how to use them in ioControl, see Opto 22 form #1410, *Tutorial: PID with SNAP Ultimate I/O Systems*.

Get PID Gain
Set PID Gain
Get PID Tune Derivative
Set PID Tune Derivative
Get PID Tune Integral
Set PID Tune Integral
Get PID Input
Get PID Current Input
Set PID Input
Get PID Input Low Range
Set PID Input Low Range
Get PID Input High Range
Set PID Input High Range
Get PID Scan Time
Set PID Scan Time
Get PID Setpoint
Get PID Current Setpoint
Set PID Setpoint
Get PID Output
Set PID Output
Get PID Output High Clamp
Set PID Output High Clamp
Get PID Output Low Clamp
Set PID Output Low Clamp
Get PID Max Output Change
Set PID Max Output Change
Get PID Min Output Change
Set PID Min Output Change
Get PID Mode
Set PID Mode
Get PID Configuration Flags
Set PID Configuration Flags
Get PID Status Flags
Get PID Feed Forward
Set PID Feed Forward
Get PID Feed Forward Gain
Set PID Feed Forward Gain
Get PID Forced Output When Input Over Range
Set PID Forced Output When Input Over Range
Get PID Forced Output When Input Under Range
Set PID Forced Output When Input Under Range

What is a PID?

A proportional integral derivative (PID) control system (often referred to as a PID loop) monitors a process variable, compares the variable's current value to a desired value (a setpoint), and calculates an output to correct error between the setpoint and the variable. Because the calculation is complex, it is done by a mathematical formula that is adjusted (tuned) for each PID loop. The mathematical formulas vary, but all PID systems share these fundamental concepts:

- They evaluate a process variable against its setpoint.
- They control an output to correct the process variable.
- The output comprises proportional, integral, and derivative calculations.
- The effect of proportional, integral, and derivative calculations is modified by user-determined P, I, and D constants.
- The P, I, and D constants need to be tuned for each system.

PID Loops on I/O Units

Analog/digital SNAP Ultimate I/O units provide 32 PID loops per I/O unit; SNAP Ethernet I/O units provide 16 PID loops per unit. Because PIDs run on the I/O side of the SNAP Ultimate brain, not on the control side, these PIDs will keep running on the Ultimate I/O unit even if the ioControl strategy stops.

In ioControl, you can configure each of the PID loops with unique settings for a large number of parameters. For a simple PID loop, you must configure at least the following:

- Input (the process variable being monitored)
- Setpoint (the desired value)
- Output (the I/O point that effects change in the system)
- Scan time (how often the input is sampled)
- PID algorithm used (four algorithms are available; see "Algorithm Choices (PID—Ethernet)" on page 10-60.
- Valid range for input
- Upper and lower clamps for output
- Minimum and maximum change for output

You can also configure the following parameters if necessary:

- Forced output value or use of manual mode if input goes out of range
- Feed forward gain
- Square root of input

In the SNAP Ultimate and SNAP Ethernet PIDs, the derivative is applied only to the process variable (the input) and not to the setpoint. This means you can change the setpoint without causing spikes in the derivative term. These PIDs also prevent integral windup by back calculating the integral without the derivative term. The feed forward term ("bias") is added before output clamping and has a tuning factor.

If desired, you can cascade PIDs by simply using the Output Point of one PID loop as the Input Point for another.

Algorithm Choices (PID—Ethernet)

When you configure a PID loop in ioControl, choose one of the following algorithms:

- Velocity
- ISA
- Parallel
- Interacting

The ISA, Parallel and Interacting algorithms are functionally equivalent; the only difference is the way the tuning constants are factored. The identical and differing equations for all algorithms are shown in the following sections.

Key to Terms Used in Equations

PV	Process variable; the input to the PID
SP	Setpoint
InLo, InHi	Range of the input
OutLo, OutHi	Range of the output
Gain	Proportional tuning parameter. Unitless. May be negative.
TuneI	Integral tuning parameter. In units of 1 seconds. Increasing magnitude increases influence on output.

TuneD	Derivative tuning parameter. In units of seconds. Increasing magnitude increases influence on output.
Output	Output from the PID
Err_1	The Error (PV – SP) from the previous scan
Integral	Integrator. Anti-windup is applied after the output is determined to be within bounds.
PvIn_1, PvIn_2	PV from the previous scan and the scan before that.
ScanTime	Actual scan time (time since previous scan)

Equations Common to All Algorithms

```
Err = PV - SP
Span = (OutHi - OutLo) / (InHi - InLo)
Output = Output + FeedForward * TuneFF
```

Velocity Algorithm

The velocity algorithm is similar to the algorithm used in OptoControl for Mistic I/O except that the derivative does not act on setpoint changes.

```
TermP = ( Err - Err_1 )
TermI = TuneI * ScanTime * Err
TermD = TuneD / ScanTime * ( PvIn - 2 * PvIn_1 + PvIn_2 )
ΔOutput = Span * Gain * ( ΔTermP + ΔTermI + ΔTermD )
```

Non-velocity Algorithms

These equations were derived from the article “A Comparison of PID Control Algorithms” by John P. Gerry in *Control Engineering* (March 1987). These three equations are the same except for the tuning coefficients; converting from one equation to another is merely a matter of converting the tuning coefficients.

Equations common to all but the velocity algorithm:

```
Integral += Err
TermP = Err
TermI = TuneI * ScanTime * Integral
TermD = TuneD / ScanTime * ( PvIn - PvIn_1 )
```

“Ideal” or ISA Algorithm:

```
Output = Span * Gain * ( TermP + TermI + TermD )
```

"Parallel" Algorithm:

```
Output = Span * ( Gain * TermP + TermI + TermD )
```

"Interacting" Algorithm:

```
Output = Span * Gain * ( TermP + TermI ) * ( 1 + TermD )
```

PID—Mistic Commands

The following commands are used for PID loops running on mistic I/O units only. For PID loops on Ethernet-based I/O units, see "PID—Ethernet Commands" on page 10-58.

Get Mistic PID P Term
Set Mistic PID P Term
Get Mistic PID I Term
Set Mistic PID I Term
Get Mistic PID D Term
Set Mistic PID D Term
Get Mistic PID Input
Set Mistic PID Input
Get Mistic PID Setpoint
Set Mistic PID Setpoint
Get Mistic PID Output
Get Mistic PID Output Rate of Change
Set Mistic PID Output Rate of Change
Get Mistic PID Scan Rate
Set Mistic PID Scan Rate
Get Mistic PID Mode
Set Mistic PID Mode to Auto
Set Mistic PID Mode to Manual
Get Mistic PID Control Word
Set Mistic PID Control Word
Clamp Mistic PID Output
Clamp Mistic PID Setpoint
Disable Mistic PID Output
Disable Mistic PID Output Tracking in Manual Mode
Disable Mistic PID Setpoint Tracking in Manual Mode
Enable Mistic PID Output
Enable Mistic PID Output Tracking in Manual Mode
Enable Mistic PID Setpoint Tracking in Manual Mode

What is a PID?

A proportional integral derivative (PID) control system (often referred to as a PID loop) monitors a process variable, compares the variable's current value to a desired value (a setpoint), and calculates an output to correct error between the setpoint and the variable. Because the calculation is complex, it is done by a mathematical formula that is adjusted (tuned) for each PID loop. The mathematical formulas vary, but all PID systems share these fundamental concepts:

- They evaluate a process variable against its setpoint.
- They control an output to correct the process variable.
- The output comprises proportional, integral, and derivative calculations.
- The effect of proportional, integral, and derivative calculations is modified by user-determined P, I, and D constants.
- The P, I, and D constants need to be tuned for each system.

Using PIDs on mistic I/O Units

Eight PID loops are available per I/O unit. The PID algorithm used with mistic protocol brains, such as the serial B3000 and the G4A8R, is the velocity PID algorithm. It is an interacting type with a reverse output.

- *Interacting* means that the gain is distributed to each term in the equation. Therefore, if you double the gain, you also double the integral and derivative terms.
- *Reverse output* means that the output increases as the input decreases. The reverse output mode is used for "pump-up" control, such as maintaining level, pressure, and flow as well as heating.

For cooling or "pump-down" control, direct output is required. To switch to direct, simply reverse the sign of the gain. For example, a gain of 1.28 would become -1.28. Note that this is not negative gain. The minus sign only serves to change the type of PID output from reverse to direct.

This velocity PID algorithm (also referred to as the incremental PID algorithm) is inherently "anti-windup" since it has no summation in the integral term to saturate. The algorithm is described on pages 160–162 of the book *Microprocessors in Instruments and Control* by Robert J. Bibbero, published by John Wiley and Sons.

Velocity PID Equation (PID—mistic)

Change in output = **Gain** *
[(Error – Last Error) +
(**Integral** * Time * Error) +
{(**Derivative**/Time) * (Error – (2 * Last Error) + Oldest Error)}]

where:

- Error is (Setpoint – Input) in Engineering Units
- Time is (Scan Rate/60), which results in time in minutes

All values are in Engineering Units.

The change in output calculated by this formula is added to the existing PID output. If the input span and the output span are different, the change is normalized and then added to the output. This is accomplished by converting the change to a percentage of the input span. The same percentage of output span is then added to the output.

Gain (P)

For those familiar with the term "proportional band," gain is simply the inverse. Gain acts directly on the change in error since the last scan. (Error is the setpoint minus the input value in engineering units.) Therefore, in the case of steady-state error (that is, change in error = 0), gain alone has no effect on the output. For this reason, gain cannot be used alone. Gain is also used as a multiplier on the integral and derivative.

The velocity PID algorithm uses gain much as it is used in the Honeywell "type A" PID and the Bailey "error input" type PID. Higher gain results in increased output change. Too much gain results in output oscillation. Too little gain results in very slow performance.

Integral (I)

This term acts only on the current error. It is used to reduce the current error to zero. Note that during steady-state conditions, integral multiplied by current error multiplied by gain is the only thing affecting the output. The larger the integral value, the larger the change in output.

A positive integral value is required. Integral that is too low will result in undershoot. Integral that is too high will result in overshoot.

Derivative (D)

This term acts only on the change in slope of the input signal. Its purpose is to anticipate where the input will be on the next scan based on a change in the rate of change of the input value. In other words, it changes the output as the input gets near the setpoint to prevent overshooting or undershooting.

Derivative is used in "feed forward" applications and in systems where the loop dead time is long. Its action type is unlimited (that is, it has no filtering). If the input signal is noisy and the derivative value is greater than zero, the input value must be filtered. See "Input Filtering" on page 10-65 for details. If the slope of the input signal has remained unchanged for the last two scans, the derivative has no effect.

Judging by the change in direction of the input, the derivative contributes an appropriate value to the output that is consistent with where the input will be at the next scan if it continues at its current rate of change.

The derivative is very useful in loops with a long dead time and long time constants. To disable it, set it to zero.

Integral-Derivative Interaction

Integral and derivative can try to move the output in opposite directions. When this is the case, the derivative should be large enough to overcome the integral. Since the derivative is "looking ahead" based on the change in slope, it has a bigger picture than the integral does.

This interaction can be observed when the input is below the setpoint and is rising fast. The integral tries to increase the output (which only makes things worse), while the derivative tries to decrease the output. The derivative does this because at the current rate of change of the input, there will be an input overshoot if the output is increased. Therefore, the derivative needs to be large enough to counteract the integral when necessary.

Configuration Tips (PID—mistic)

Gain The gain value must not be zero. If input engineering units are negative, the output may move in the direction opposite from the one desired. If so, reverse the sign of the gain.

Integral The integral is required and must be greater than zero.

Input The input must not be bipolar. An input range of -10 to +10, for example, will not work. Values such as -300 to -100, -100 to 0, and 0 to 100 are acceptable. If an application has a bipolar input range, it will have to be rescaled to a generic range, such as 0 to 100. The setpoint range will then have to match this generic input range.

Setting the output lower and upper clamps Setting clamps is particularly important if the device controlled by the output signal has “dead areas” at either end. For example, say the output is scaled 0–10. It is connected to a valve that begins to open at 1.25 and is “effectively” fully open at 5.75 (even though it may only be 70% open). Set Lower Clamp to 1.2 (valve closed) and Upper Clamp to 5.75 (valve effectively fully open). This prevents reset windup, potentially resulting in dramatically improved control when the output value has reached either limit and has to suddenly reverse direction.

Setting the maximum change rate of the output The Max Change Rate can be ignored, since it defaults to 100% per scan.To limit the output rate of change, set Max Change Rate to 10%to start. This setting would limit the output rate of change to 100% in 10 scan-rate periods.

Output The output can be preset or changed at any time by an operator or by the program. For example, if the output should start at 40% whenever the system is activated, simply set the PID output (or the analog channel output) to this value under program control.

Manual Mode The factory default causes the setpoint to track the input when the PID is in manual mode, which means that the setpoint will be altered when in manual mode. If you don’t want the setpoint to be altered when in manual mode, disable the setpoint track output feature so that when the PID is in manual mode, the setpoint will not be changed.

Input Filtering If the input signal is noisy, you may want to filter it. To do so, follow these steps:

1. Use the command Set Analog Filter Weight, specifying the appropriate analog input channel. Use a filter weight value of less than 10 times the scan rate. Otherwise, the loop cannot be tuned.

2. Configure the PID loop to use the average/filtered value.

3. You can store the configuration to EEPROM or Flash memory to save the filter weight and the input type (current or average). This can be helpful when reenabling an I/O unit after a loss of communication.

Tuning Guidelines (PID—mistic)

Setting the Scan Rate

The scan rate should be set as fast as possible or as fast as the controlled equipment will allow, unless there are rare and unusual circumstances. Setting the scan rate to be longer than the dead time will result in a PID controller that is so sluggish that it cannot adequately respond to disturbances, and setpoint changes will be extremely slow.

There are, however, exceptions to this rule. If the output of the PID is implemented on equipment that cannot handle fast scan changes, set the PID scan loop to as fast as the equipment can handle. Most control valves, which are very commonly controlled by PID loops, can handle 0.1 second scan rates just fine. However, there are definitely types of equipment that can't handle this. One example would be big gates at big dams. Moving one of these gates is a major event that takes a long time, so one of the control goals is to minimize gate movement. For the flow controllers on these gates, it may be necessary to set the scan time several times longer than the dead time. This allows reaching the setpoint with fewer moves of the gate.

The terms "aggressive" and "conservative" are extremely subjective and depend on the process. "Aggressive" tuning can be thought of as tuning where the speed of reaching the setpoint is the primary concern, and overshoot is tolerated to gain fast response. "Conservative" tuning can be thought of as tuning required in a system where overshoot is not tolerated, and speed of response is sacrificed to prevent overshoot.

The tuning suggestions given here are to achieve the fastest response with no overshoot. This would be on the "aggressive" side of "conservative" tuning. Tuning rules found in control textbooks such as Ziegler-Nichols typically advocate "quarter amplitude decay," which means an overshoot that results in an undershoot that is 1/4 the amplitude of the overshoot and then an overshoot that is 1/4 the amplitude of the undershoot, etc. until it stabilizes on the setpoint. This kind of tuning could be considered "very aggressive."

Determining the Loop Dead Time

To determine the dead time, put the PID output in manual mode, then set the output somewhere around midrange. After the loop has achieved a steady state, change the output by at least 10% of its span. Measure the time (in seconds) that it takes the input to start responding to the change. This is the dead time.

Tuning

The tuning guidelines below are followed by a series of graphs showing the effects of implementing various multiples of the "optimal" gain and integral. The "optimal" gain and integral are multiplied by 2 (too high) and 0.5 (too low), and every combination of these tuning parameters is shown on the graphs. Comparing actual PID loop performance with these graphs can usually identify what adjustments are necessary to improve the tuning of the actual PID loop. It is important to use a graphical tool, like ioDisplay, to assist in the tuning process. (Note that the graphical PID tuner in ioControl cannot be used for tuning mistic PID loops.)

These graphs and guidelines are just generalizations. They won't be valid in all possible cases; they are just a guide to help.

These tuning guidelines can be used both to solve tuning problems in an existing loop or as a help to start tuning a new loop.

IMPORTANT NOTE: Textbook tuning rules, such as Ziegler-Nichols tuning methods, DO NOT work for tuning the velocity PID algorithm.

Solving Tuning Problems

Oscillations Oscillations can be caused either by gain that is too high or integral that is too high. If the process variable oscillates below the setpoint, it is probably caused by the gain being too high. If it oscillates at the setpoint, it is not possible to know by looking at the graphs which tuning parameter is causing the problem. Try cutting either the gain or integral, but not both at the same time, to find out which one is causing the problem.

Overshoot Overshoot is usually caused by the integral being too high. Gain that is too high can also cause overshoot, but that is usually in conjunction with the integral being too high.

Any PID loop can be made to not overshoot and not oscillate if the gain and integral are set low enough, but the response will be slow.

Performance There is a limit on how fast a good, stable response can be. The middle chart is the best that can be done with no overshoot and no oscillation. The ones with the gain and integral too high move toward the setpoint faster, but they overshoot and oscillate. There will be a point that is the best the PID loop can be tuned, and it will not be possible to get a faster stable response. There are trade-offs between having a fast response and having a stable PID loop that does not overshoot the setpoint.

Starting the Tuning Process for a New PID Loop

A simple and safe method is to start out at a very low gain and integral (lower left chart—see page 10-68). Increase the gain without changing the integral until the fastest response is achieved without oscillation. It still won't reach the setpoint, but the key here is getting a fast response that doesn't oscillate (middle left graph). Now leave the gain alone and increase the integral until it reaches the setpoint without oscillating (middle graph). This completes the tuning.

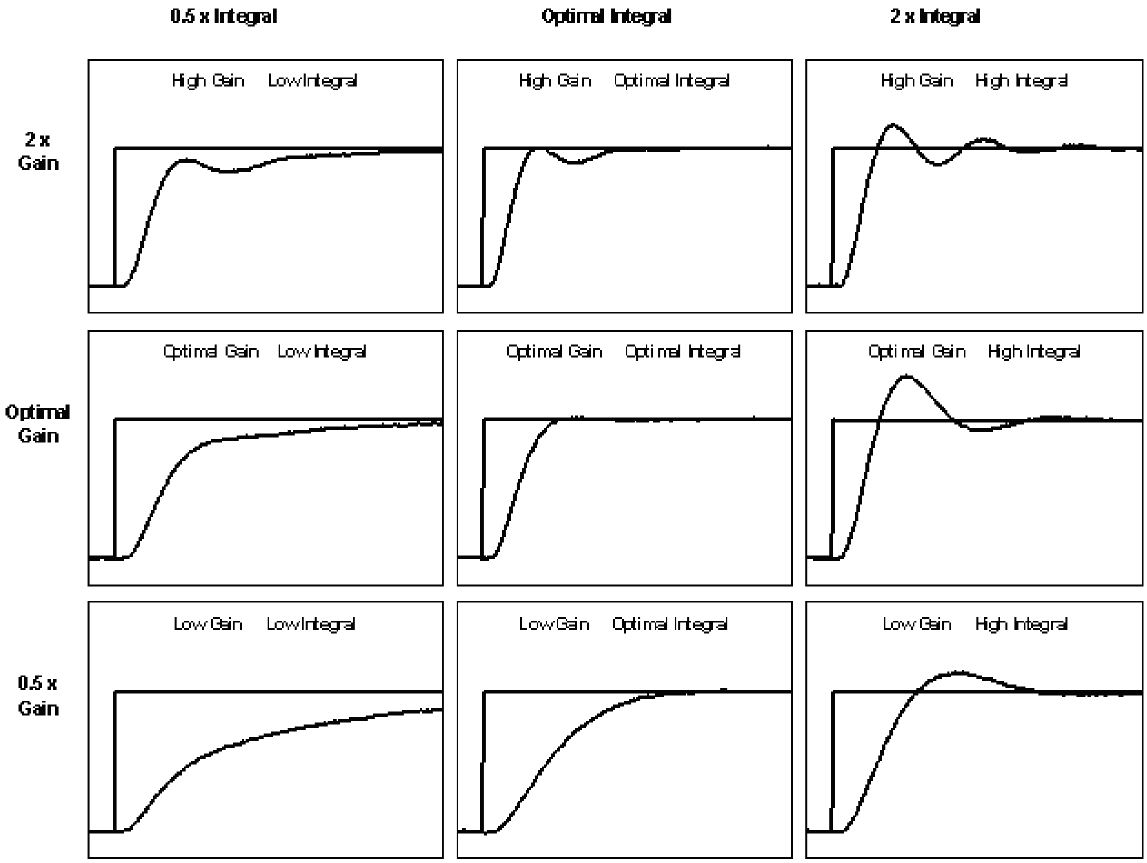
When increasing the gain and integral, it is fastest to just keep doubling them. When the process variable starts oscillating, then make smaller gain and integral changes.

For example, start out with a gain of 0.1 and an integral of 0.1. Next try a gain of 0.2 while keeping the integral at 0.1; then a gain of 0.4, then 0.8, then 1.6, then 3.2. If the PID loop starts oscillating with a gain of 3.2, then try a gain of 2.0 or something in the middle between 1.6 and 3.2. Then make smaller changes until the best gain is found. Suppose the best gain was 2.3; the next step is to keep the gain at 2.3, and then change the integral to 0.2, to 0.4, and then to 0.8, and so on, until the best integral is found.

Derivative

Tuning the derivative term is not addressed in these graphs because most PID loops are fine without it. Derivative in the Opto 22 implementation of the velocity PID algorithm works fine for disturbance rejection; the derivative should be kept very, very low. However, using derivative does not work well for setpoint changes because it will cause spikes in the output. This is because the derivative term of the velocity PID algorithm is calculated based on the error, which is the difference between the setpoint and the process variable. If the setpoint changes, then instantly a jump in error occurs and results in a jump in output. Therefore, it is best to use a derivative term of 0 when cascading PID loops.

Tuning Graphs (PID—mistic)



Simulation Commands

The following commands are used for simulation and program testing:

Communication to All I/O Points Enabled?
Communication to All I/O Units Enabled?
Disable Communication to Point
Disable Communication to All I/O Points
Disable Communication to I/O Unit
Disable Communication to All I/O Units
Disable Communication to PID Loop
Enable Communication to Point
Enable Communication to All I/O Points
Enable Communication to I/O Unit
Enable Communication to All I/O Units
Enable Communication to PID Loop
I/O Point Communication Enabled?
I/O Unit Communication Enabled?
PID Loop Communication Enabled?
IVAL Turn Off
IVAL Turn On
IVAL Set Analog Point
IVAL Set Counter
IVAL Set Off-Latch
IVAL Set On-Latch
Disable Communication to Mistic PID Loop[1,2]
Enable Communication to Mistic PID Loop[1,2]
Mistic PID Loop Communication Enabled?[1,2]
Disable Communication to Event/Reaction[1,2]
Disable Event/Reaction Group[1,2]
Enable Communication to Event/Reaction[1,2]
Enable Event/Reaction Group[1,2]
Event/Reaction Communication Enabled?[1,2]
Event/Reaction Group Communication Enabled?[1,2]
IVAL Set I/O Unit from MOMO Masks
IVAL Set Off-Pulse[1]
IVAL Set On-Pulse[1]
IVAL Set Off-Totalizer[1,2]
IVAL Set On-Totalizer[1,2]
IVAL Set TPO Percent[1]
IVAL Set TPO Period[1]
IVAL Set Period[1,2]
IVAL Set Frequency[1,2]
IVAL Set Mistic PID Control Word[1,2]
IVAL Set Mistic PID Process Term[1,2]

The Disable commands disconnect the strategy from the real-world device, so that it can be tested without affecting field devices. While the real-world devices are disabled (or if they don't exist) the IVAL commands can be used for testing and simulation. For details on individual commands, see the *ioControl Command Reference* or online Help.

[1] ioControl Professional only
[2] mistic I/O units only

CHAPTER 11

Using OptoScript

Introduction

This chapter shows you how to create and use OptoScript, an optional programming language that can simplify certain types of operations in ioControl. Modeled after computer languages such as C and Pascal, OptoScript code gives you an alternative to using standard ioControl commands.

You will find OptoScript easy to use if you already have computer programming experience. Beginning programmers may also want to try it for control operations involving extensive math calculations, string handling, or complex loops and conditions.

This chapter assumes that you have some programming experience. Experienced programmers may want to see "Notes to Experienced Programmers" on page F-6.

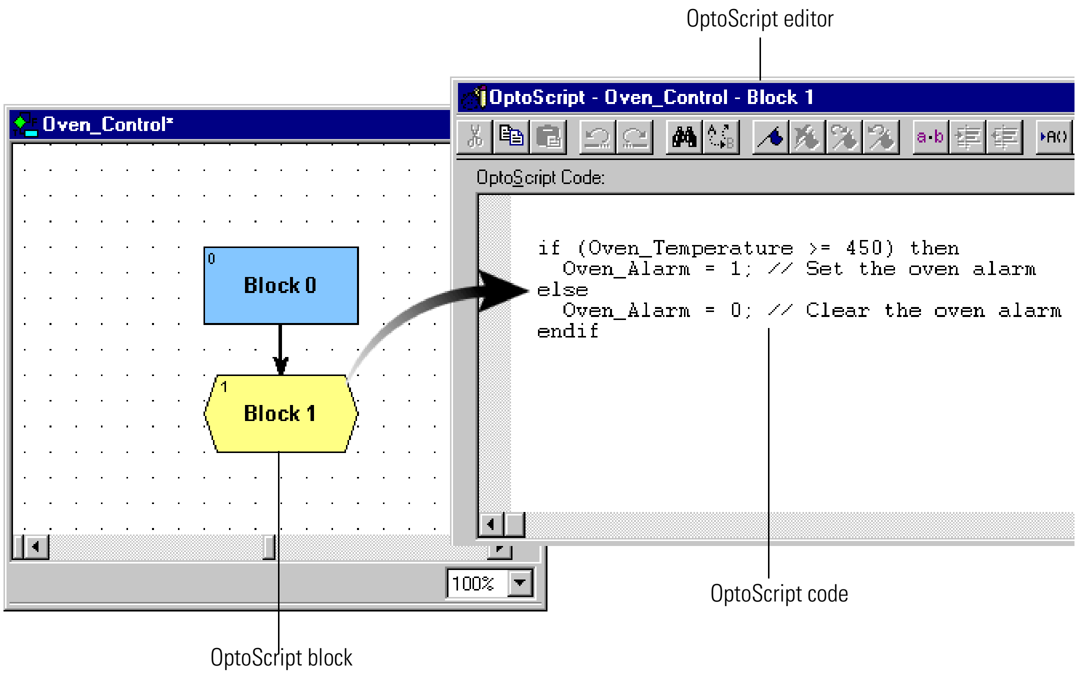
In this Chapter

About OptoScript

OptoScript is a procedural type of computer language similar to Pascal, C, or BASIC. It can be used within any ioControl strategy or subroutine to replace or supplement standard ioControl commands. It does not add new functions, but offers an alternative method within ioControl's flowcharting environment to simplify some common programming tasks.

OptoScript code cannot be mixed with commands in action or Condition Blocks; it is used in its own hexagonal flowchart block.

The following figure shows an example of an OptoScript flowchart block and its contents:



```
if (Oven_Temperature >= 450) then
  Oven_Alarm = 1; // Set the oven alarm
else
  Oven_Alarm = 0; // Clear the oven alarm
endif
```

When To Use OptoScript

You'll want to use OptoScript for some common programming tasks that can be more difficult to do using standard ioControl commands than using a procedural language. Extensive math calculations or complex loops, for example, can be done with standard commands but take up a lot of space on a flowchart.

When you use OptoScript, however, be aware that it is not self-documenting. Make sure you frequently use comments to explain what the code does, so that when you come back to it a year later—or when someone who is not as familiar with the code or the strategy must change it—it can be easily interpreted.

This section shows examples of using OptoScript:

- for math expressions
- for string handling
- for complex loops
- for case statements
- for conditions
- for combining math expressions, loops, and conditions.

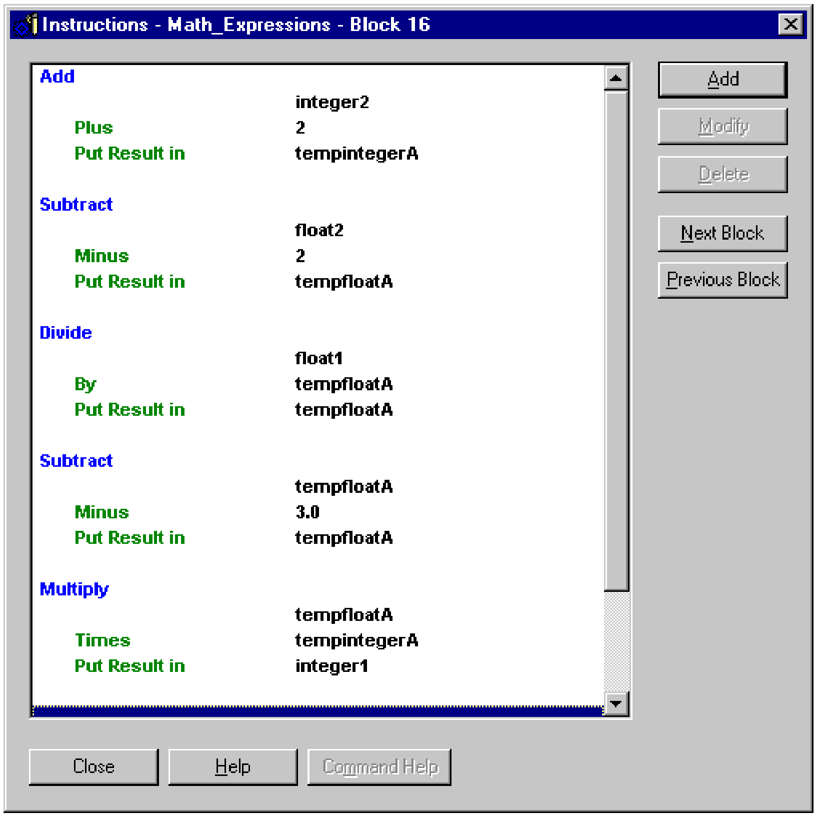
For Math Expressions

OptoScript is especially useful for mathematical computations. Math expressions are simpler and easier, and many of them are built right into the language, instead of requiring commands such as Add or Multiply. OptoScript has no limitations on the number of parentheses you can use in math expressions.

Here's an example of a mathematical expression in OptoScript:

```
integer1 = (integer2 + 2) * (float1 / (float2 - 2) - 3);
```

To accomplish the same computation using standard ioControl commands, you would need to create at least two intermediate variables and use five instructions, as shown below.



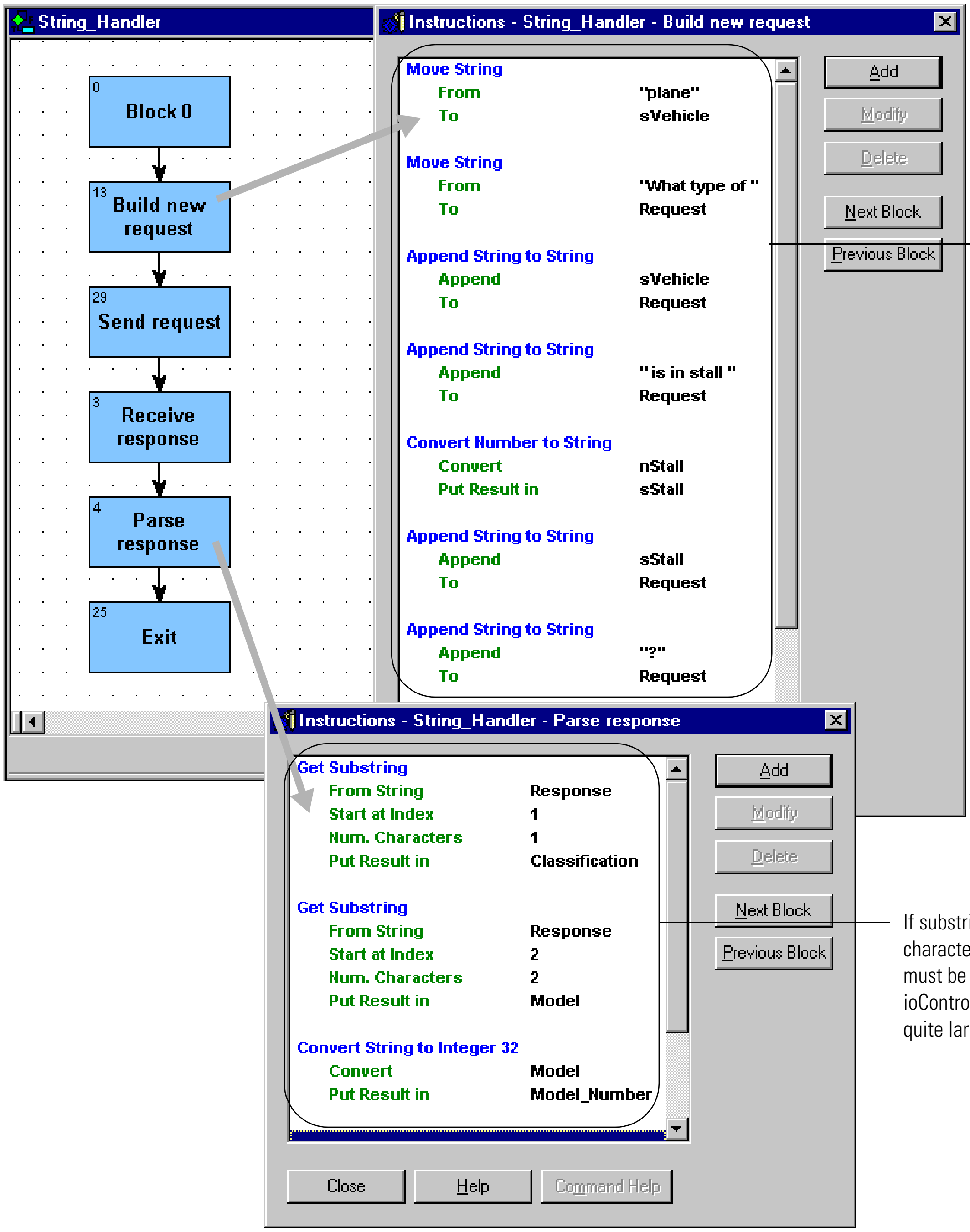
```
Add
                        integer2
   Plus                 2
   Put Result in        tempintegerA

Subtract
                        float2
   Minus                2
   Put Result in        tempfloatA

Divide
                        float1
   By                   tempfloatA
   Put Result in        tempfloatA

Subtract
                        tempfloatA
   Minus                3.0
   Put Result in        tempfloatA

Multiply
                        tempfloatA
   Times                tempintegerA
   Put Result in        integer1
```

As you can see, the OptoScript version of this math expression is not only simpler to create, but also easier to understand once created.

For String Handling

If your strategy transmits and receives serial data, you will want to try using OptoScript code. In standard ioControl, forming and parsing (decoding) serial data can take several blocks. In OptoScript, string handling can be easier.

The following figure shows a flowchart designed to send the string request, “What type of plane?” and parse the response, “F14,” into a classification (F) and a model number (14). Compare these blocks and instructions with the ones on the following page, done in OptoScript.



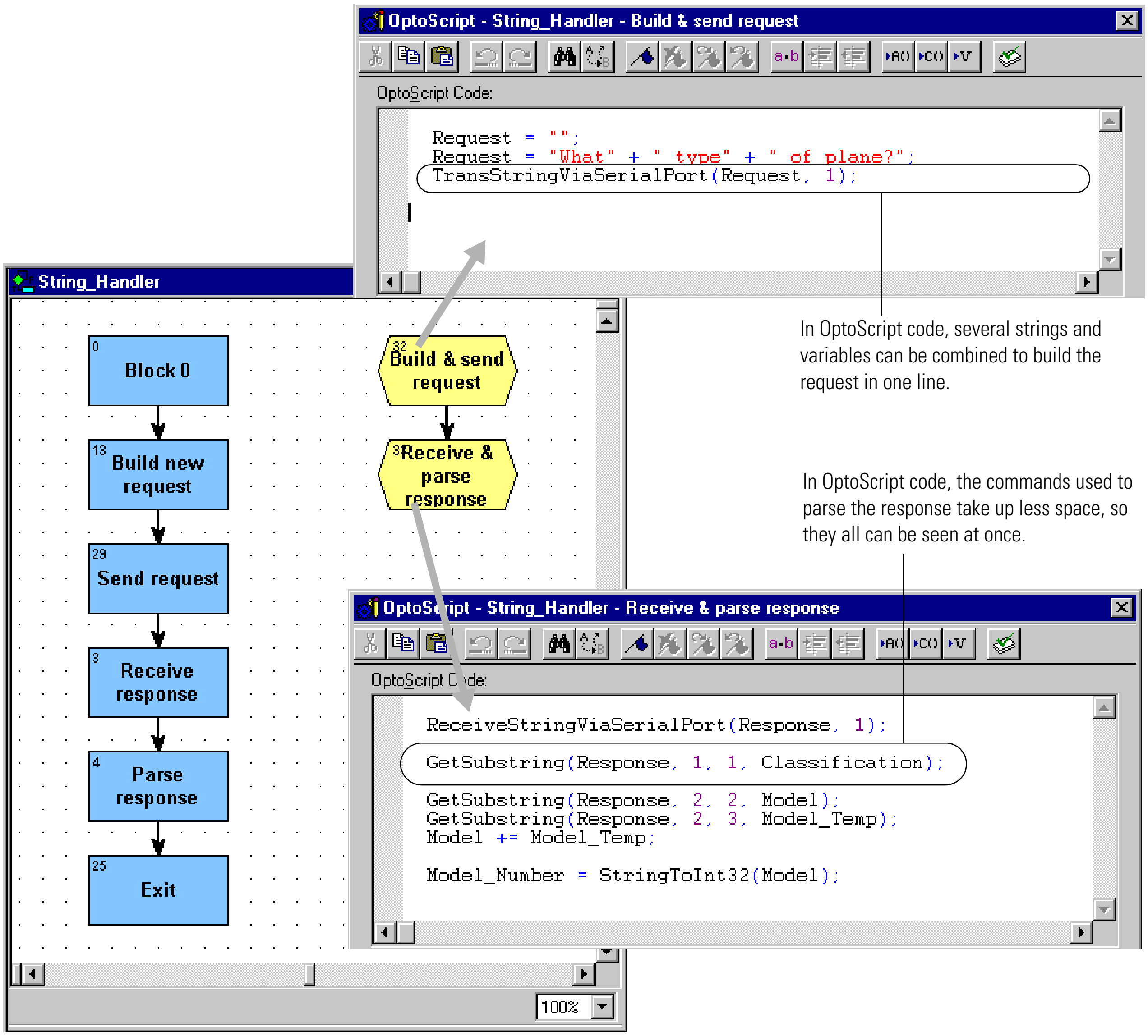
```
Move String
    From                    "plane"
    To                      sVehicle

Move String
    From                    "What type of "
    To                      Request

Append String to String
    Append                  sVehicle
    To                      Request

Append String to String
    Append                  " is in stall "
    To                      Request

Convert Number to String
    Convert                 nStall
    Put Result in           sStall

Append String to String
    Append                  sStall
    To                      Request

Append String to String
    Append                  "?"
    To                      Request
```

Building a string using standard ioControl can require several commands.

```
Get Substring
    From String             Response
    Start at Index          1
    Num. Characters         1
    Put Result in           Classification

Get Substring
    From String             Response
    Start at Index          2
    Num. Characters         2
    Put Result in           Model

Convert String to Integer 32
    Convert                 Model
    Put Result in           Model_Number
```

If substrings or individual characters within a string must be handled, a standard ioControl block can become quite large.

The OptoScript version of the String_Handler flowchart is more compact. The string request can be built more easily, and parsing the response takes up much less space. If you handle more complex serial data than in the String_Handler example, you will find OptoScript code even more useful.



```
Request = "";
Request = "What" + " type" + " of plane?";
TransStringViaSerialPort(Request, 1);
```

In OptoScript code, several strings and variables can be combined to build the request in one line.

In OptoScript code, the commands used to parse the response take up less space, so they all can be seen at once.

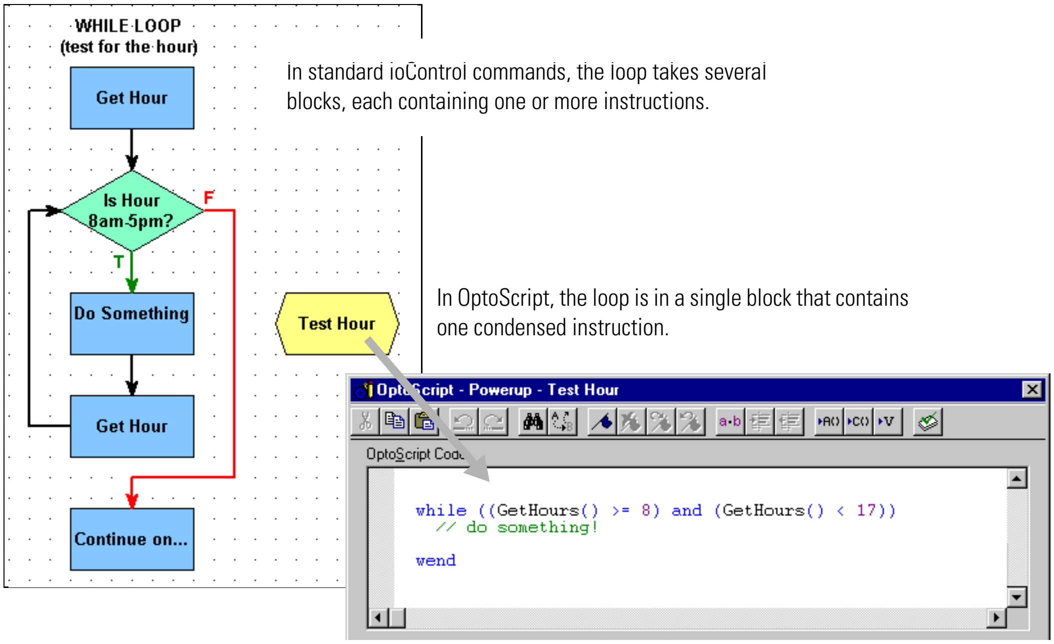
```
ReceiveStringViaSerialPort(Response, 1);
GetSubstring(Response, 1, 1, Classification);
GetSubstring(Response, 2, 2, Model);
GetSubstring(Response, 2, 3, Model_Temp);
Model += Model_Temp;

Model_Number = StringToInt32(Model);
```

For Complex Loops

Strategies that use complex loops—for example, to repeat an operation while a condition remains true—are easier to create and take up less space in a flowchart when done in OptoScript. *While* loops, *repeat* loops, and *for* loops are all available.

- **While loops** repeat a process while a test is true (the test comes at the beginning of the process).
- **Repeat loops** repeat a process until a test is false (the test comes at the end of the process). This kind of loop is guaranteed to execute at least once.
- **For loops** repeat a process for a specified number of times.

Below is an example of a *while* loop as it would appear in standard flowchart commands, contrasted with the way it could be handled in an OptoScript Block.



In standard ioControl commands, the loop takes several blocks, each containing one or more instructions.

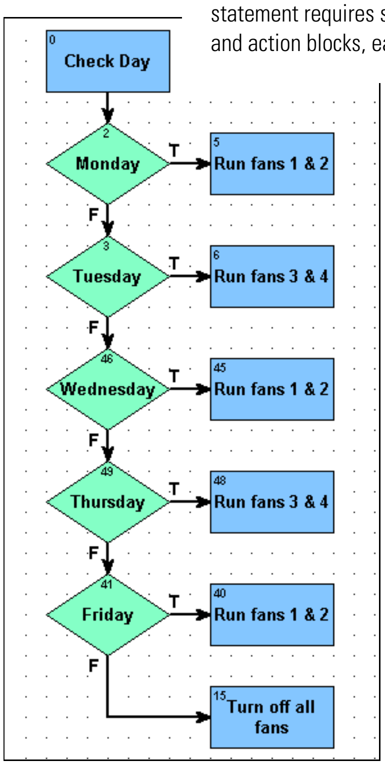
In OptoScript, the loop is in a single block that contains one condensed instruction.

```
while ((GetHours() >= 8) and (GetHours() < 17))
  // do something!

wend
```

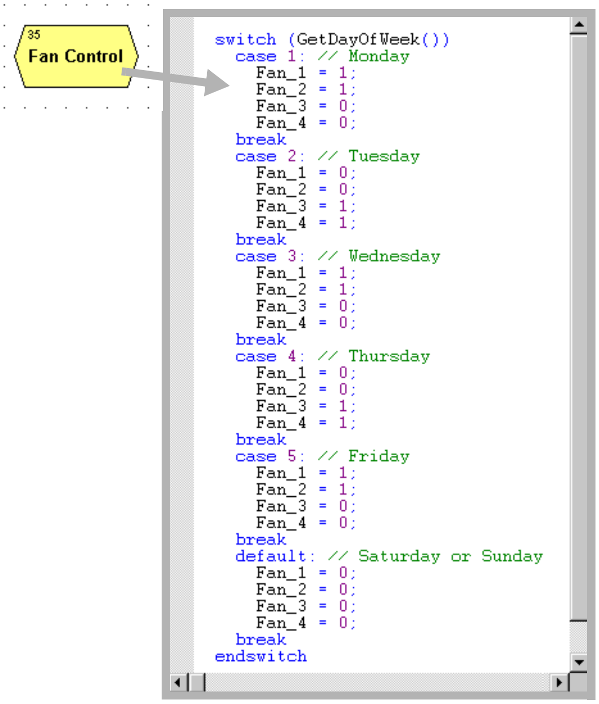
For Case Statements

Case or *switch* statements create multiple decision points. They can also be easier to do using OptoScript. Here is an example of a case statement:

In standard ioControl commands, the case statement requires several sets of condition and action blocks, each containing commands.



In OptoScript, the code is all in one block.

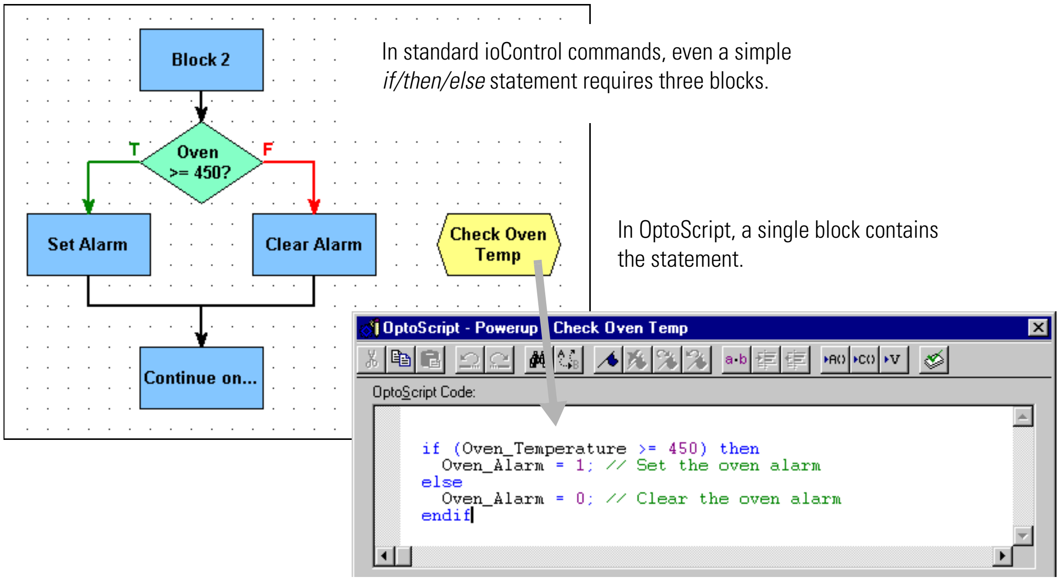


```
switch (GetDayOfWeek())
  case 1: // Monday
    Fan_1 = 1;
    Fan_2 = 1;
    Fan_3 = 0;
    Fan_4 = 0;
  break
  case 2: // Tuesday
    Fan_1 = 0;
    Fan_2 = 0;
    Fan_3 = 1;
    Fan_4 = 1;
  break
  case 3: // Wednesday
    Fan_1 = 1;
    Fan_2 = 1;
    Fan_3 = 0;
    Fan_4 = 0;
  break
  case 4: // Thursday
    Fan_1 = 0;
    Fan_2 = 0;
    Fan_3 = 1;
    Fan_4 = 1;
  break
  case 5: // Friday
    Fan_1 = 1;
    Fan_2 = 1;
    Fan_3 = 0;
    Fan_4 = 0;
  break
  default: // Saturday or Sunday
    Fan_1 = 0;
    Fan_2 = 0;
    Fan_3 = 0;
    Fan_4 = 0;
  break
endswitch
```

Using OptoScript for case statements saves space in the flowchart and lets you see all the possible cases in one dialog box.

For Conditions

Like loops and case statements, conditions can be simpler when done in OptoScript code. *If/then*, *if/then/else*, and *if/then/elseif* statements can all be mixed and nested as needed. Here's an example of a simple *if/then/else* statement as it could be done in standard ioControl commands and in OptoScript:

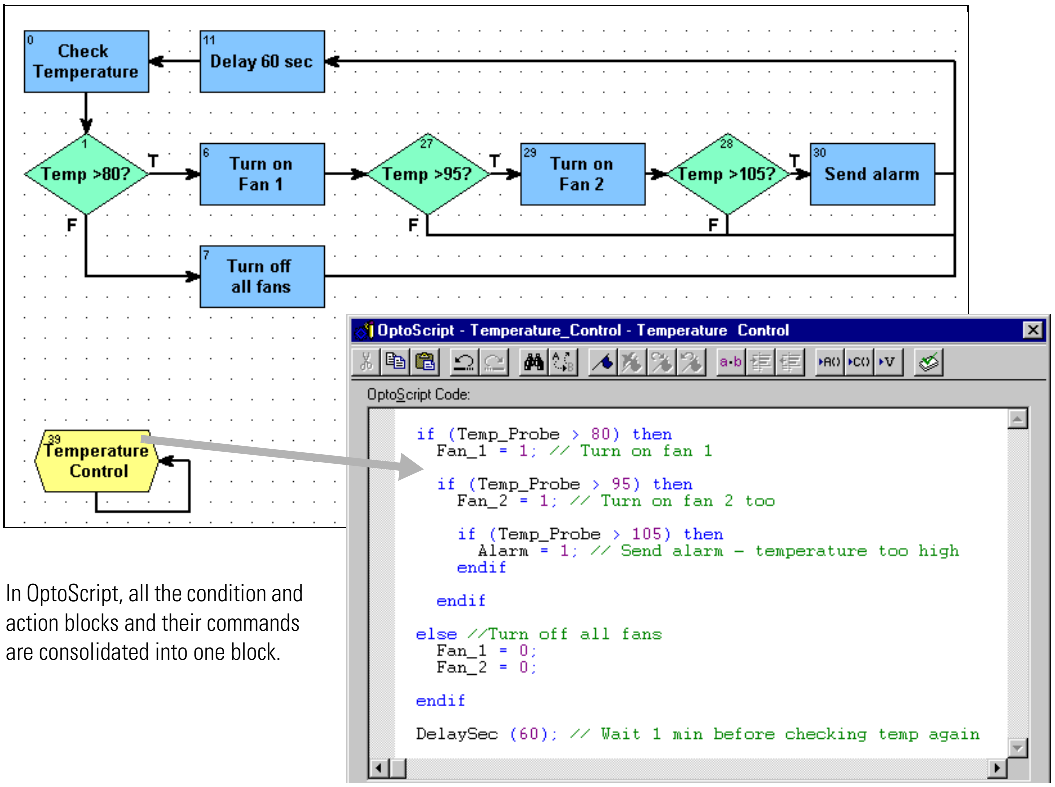


In standard ioControl commands, even a simple *if/then/else* statement requires three blocks.

In OptoScript, a single block contains the statement.

```
if (Oven_Temperature >= 450) then
  Oven_Alarm = 1; // Set the oven alarm
else
  Oven_Alarm = 0; // Clear the oven alarm
endif
```

OptoScript is even more useful for more complex conditions, such as the following:

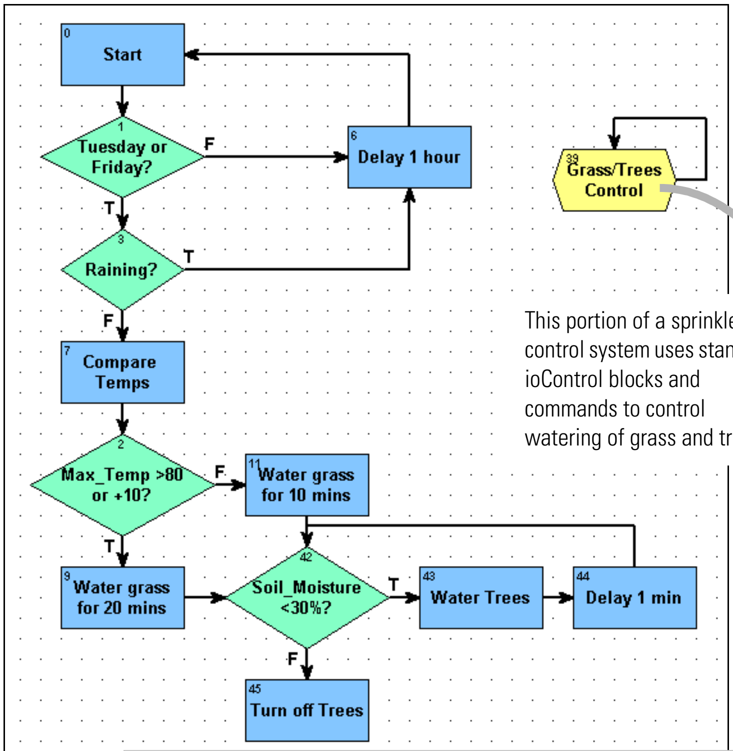


In OptoScript, all the condition and action blocks and their commands are consolidated into one block.

```
if (Temp_Probe > 80) then
  Fan_1 = 1; // Turn on fan 1

  if (Temp_Probe > 95) then
    Fan_2 = 1; // Turn on fan 2 too

    if (Temp_Probe > 105) then
      Alarm = 1; // Send alarm - temperature too high
    endif

  endif

else //Turn off all fans
  Fan_1 = 0;
  Fan_2 = 0;

endif

DelaySec (60); // Wait 1 min before checking temp again
```

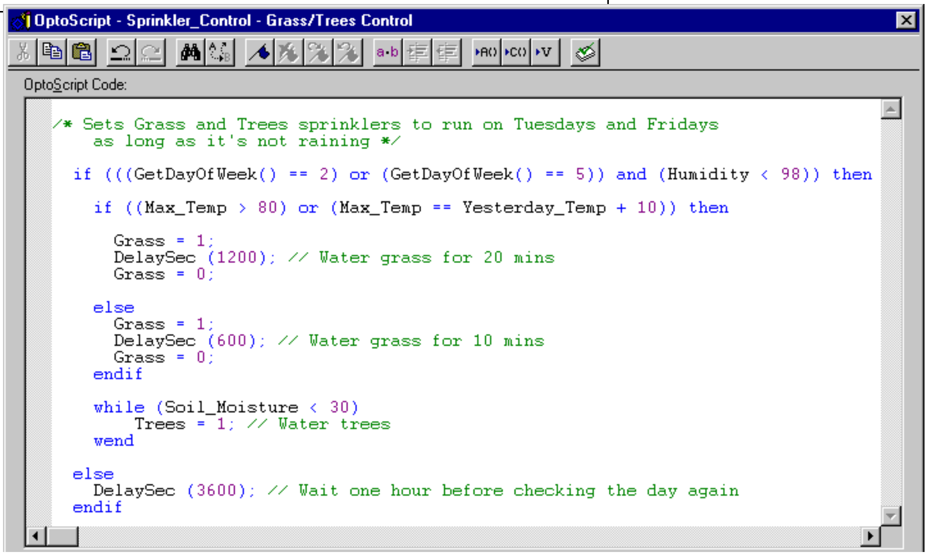
For Combining Expressions, Operators, and Conditions

The real power of OptoScript can be seen in complex operations.



This portion of a sprinkler control system uses standard ioControl blocks and commands to control watering of grass and trees.

The OptoScript version of Grass/Trees Control handles the loops, conditions, and operators easily in a single block.



```
/* Sets Grass and Trees sprinklers to run on Tuesdays and Fridays
   as long as it's not raining */

 if (((GetDayOfWeek() == 2) or (GetDayOfWeek() == 5)) and (Humidity < 98)) then

   if ((Max_Temp > 80) or (Max_Temp == Yesterday_Temp + 10)) then

     Grass = 1;
     DelaySec (1200); // Water grass for 20 mins
     Grass = 0;

   else
     Grass = 1;
     DelaySec (600); // Water grass for 10 mins
     Grass = 0;
   endif

   while (Soil_Moisture < 30)
       Trees = 1; // Water trees
   wend

 else
   DelaySec (3600); // Wait one hour before checking the day again
 endif
```

Generally speaking, the more complex the combination of math expressions, logical and comparison operators, loops, and conditions, the more convenient it is to use OptoScript code rather than standard blocks and commands.

OptoScript Functions and Commands

Since functions in OptoScript are provided by commands almost identical to the standard commands in ioControl, you have the same complete range of functions. There are no additional functions for OptoScript code, and you cannot make your own functions.

Standard and OptoScript Commands

In many cases you can easily recognize OptoScript commands, because they are almost the same as standard ioControl commands. All spaces are removed from the OptoScript commands, however, and in some cases words in the command are abbreviated or left out. Commands are case sensitive. Here are some examples of the same commands in ioControl and in OptoScript:

ioControl Command	OptoScript Command
Get Counter	GetCounter
Set Down Timer Preset Value	SetDownTimerPreset
Delay (mSec)	DelayMsec
Convert Float to String	FloatToString
Get Number of Characters Waiting	GetNumCharsWaiting

Some commands are built into OptoScript functionality. Some of these have OptoScript commands and some do not; you can use either the built-in functionality or the OptoScript command, if it exists. Here are some examples:

ioControl Command	OptoScript Command	Built-In Equivalent	Example
Move		=	item1 = value
Add		+	1 + 2
Less?		<	value1 < value2
Turn On	TurnOn	= [non-zero]	digital3 = 1
Turn Off	TurnOff	= 0	digital3 = 0
Comment (Single Line)		//	// comment
Set Nth Character	SetNthCharacter		s1[5] = 'c'

See Appendix E for a table of all ioControl commands and their OptoScript equivalents. In addition, OptoScript equivalents for each command are shown in the *ioControl Command Reference* and in the online command help.

Using I/O in OptoScript

One advantage of OptoScript is that any named I/O point can be used directly, wherever a numeric variable can be used, rather than requiring a variable. Digital points behave like integer variables that have only two possible states: zero (off) or non-zero (on). Analog points behave like float variables.

For example, you can turn a Digital Point off by simply assigning it a value of zero:

```
Light_Switch = 0;
```

You can turn a digital point on by assigning it any value other than zero:

```
Light_Switch = 1;
Light_Switch = -1;
Light_Switch = 486;
```

You can use I/O points directly in mathematical expressions:

```
fLimit = Pressure_Input + 50;
```

Or use them directly in control structures, for example to turn off the light if the door is closed:

```
if (not Door) then
  Light_Switch = 0;
endif
```

You can set an output based on the value of an input or a variable:

```
LED01 = Switch_A;
Proportional_Valve = fPressure_Control
```

You can use a point directly with a command:

```
fRange = GetAnalogMaxValue(Temp_Input) - GetAnalogMinValue(Temp_Input);
TurnOn(Fan_A);
IsOn(Fan_A);
```

OptoScript Syntax

Here is a sample section of OptoScript code to illustrate syntax. Indentation is not required, but is used for clarity.

```
fPressure = 300.0;

nTotal = ntTable[0] + ntTable[1] + ntTable[2];

while ((GetHours() >= 8) and (GetHours() < 17))
  Fan_A = 1;
wend

// Send alarm if oven temperature too hot.
if (Oven_Temperature >= 450) then
  Oven_Alarm = 1; // Set the oven alarm
else
  Oven_Alarm = 0; // Clear the oven alarm
endif

nCheck = GenerateChecksumOnString (0, sMessage);
nError_Block = GetIdOfBlockCausingCurrentError();
RemoveCurrentError();

sGreeting = “Hello, world!”;
nPos = FindCharacterInString('!', 0, sGreeting);
```

Each statement is followed by a semicolon.

Table elements are put in square brackets next to the table name.

Parentheses are used as separators for expressions and operators. You can use an unlimited number of parentheses.

Line comments appear on a separate line or after a statement. They are preceded by two slashes and a space. Block comments (not illustrated) are preceded by `/*` and followed by `*/`.

Parameters (arguments) for a command are listed in order within parentheses following the command. Commands that have no arguments must still include the parentheses.

An individual character can be in single quotes or in double quotes, depending on its type. A string must be in double quotes.

NOTE: Each block has only one exit point. It is not possible to use `return` *to jump out of the current block.*

More About Syntax with Commands

As noted in the previous sample, arguments for a command are listed in the parentheses following the command. Arguments are listed in order beginning with argument 1. To find out the arguments for any command, see the *ioControl Command Reference* or online command help.

```
SetDownTimerPresetValue (60.0, Minute_Timer)
         command         (argument 1, argument 2)

EnableIOUnitCausingCurrentError ()
            command              (no arguments)
```

Commands in OptoScript can be broken into two categories: procedure commands and function commands.

Procedure commands accomplish an action and return no value. Here are some examples:

```
RemoveCurrentError();
ClampInt32TableElement(10, 0, 5, x1);
```

Function commands return a value from their action, so the value can be placed somewhere. In the following examples, the value is placed in the variable at the beginning of the statement:

```
nMonth = GetMonth();
fSquare_Root = SquareRoot(99);
nPosition = FindCharacterInString('S', 0, sName);
```

When you compare these examples to the identical commands in standard ioControl, you'll notice that the returned value for the standard ioControl command is an argument. In OptoScript the returned value is not an argument, thus reducing the number of arguments by one. In the first example, the standard command Get Month has one argument, which is where the result is placed. The OptoScript command equivalent, GetMonth, has no arguments and places the result in the variable.

In most cases you will use the value a function command returns by placing it in a variable, a control structure, or a mathematical expression. Occasionally, however, you may not need to use the result. For example, the command StartChart returns a status. If you do not need to track the status, you can ignore it by not placing the result anywhere, as shown below:

```
StartChart(Fan_Control);
```

OptoScript Data Types and Variables

Unlike most procedural languages, ioControl maintains a database of all declared variables, which is shared with ioDisplay. Variables are not declared in OptoScript code, but are created (declared) within ioControl. (See Chapter 9, "Using Variables and Commands.") Variables are not declared in OptoScript because local variables are not allowed. All variables are global for the strategy (or global within a subroutine).

If you use a variable in OptoScript code that does not currently exist in the strategy, you'll receive an error message when you test compile the code and can add the variable then.

Variable Name Conventions

With OptoScript and in ioControl generally, it's a good idea to get into the habit of indicating the variable type in each variable's name. Some variable types may be obvious in the name itself, but others are not. For example, a variable named *Month* might be either a string or an integer.

An easy way to avoid this confusion is to use Hungarian notation—that is, to place letters indicating variable type at the beginning of the name. For example, *sMonth* would indicate a string; *nMonth* would indicate an integer. The following table shows suggested notation for use in ioControl:

Variable type	Letter
integer 32 variable	n
integer 32 variable used as Boolean	b
integer 32 table	nt
integer 64 variable	nn
integer 64 table	nnt
float variable	f
float table	ft
down timer	dt
up timer	ut
string variable	s
string table	st

Variable type	Letter
pointer variable	p
pointer table	pt
digital I/O unit	dio
mixed I/O unit	mio
analog input point	ai
analog output point	ao
digital input point	di
digital output point	do
chart	cht
communication handle	cmh

Using Numeric Literals

Here are examples of how to use numeric literals in OptoScript. Formats are automatically converted if they don't match the variable type. For example, if a value of 300.2 were assigned to an integer 32, the value would be converted to 300.

Decimal Integer 32 Literals assigned to variables:

```
nVariable1 = 0;
nVariable2 = 10;
nVariable3 = -123;
```

Hexadecimal Integer 32 Literals assigned to variables. Hex notation starts with 0x. Digits A–F may be upper or lower case:

```
nVariable1 = 0x0;
nVariable2 = 0x10;
nVariable3 = 0x12AB34CD;
nVariable3 = 0x12ab34cd;
```

Decimal Integer 64 Literals assigned to variables. Integer 64s have an i64 at the end:

```
dVariable1 = 0i64;
dVariable2 = 10i64;
dVariable3 = -123i64;
```

Hexadecimal Integer 64 Literals assigned to variables:

```
dVariable1 = 0x0i64;
dVariable2 = 0x10i64;
dVariable3 =
0x1234567890ABCDEFi64;
```

Float Literals assigned to variables (Float literals may use scientific notation):

```
fVariable1 = 0.0;
fVariable2 = 12.3;
fVariable3 = -123.456;
fVariable3 = -1.23456e2;
fVariable3 = -12345.6e-2;
```

Making Assignments to Numeric Variables

Values are easily assigned to variables.

Simple Integer 32 assignments:

```
n1 = 1;
n2 = n1;
```

Simple Float assignments:

```
f1 = 3.0;
f2 = f1
```

Simple Integer 64 assignments:

```
nn1 = 2i64;
nn2 = nn1;
```

Simple assignments between different data types (Types will be automatically converted to match):

```
n1 = 4.0;
nn1 = n1;
f1 = n1;
```

Using Strings

As noted in the section on syntax, a string in OptoScript must be in double quotes. An individual character can be used either as a string (in double quotes) or as an integer value representing that character in ASCII (in single quotes). When you assign a single character to a string, use double quotes to avoid a syntax error:

```
sString = "a";
```

To change a single-character integer into a string, use the `Chr()` keyword as shown below:

```
 sString = Chr('a');
  sString = Chr(97);
```

```
      n = 97;
sString = Chr(97)
```

Strings can be used in the following ways.

String literals (must be all on one line):

```
sGreeting = "Hello, world!"
```

When you use the Chr() keyword to assign a character value to a string variable, you can either quote a character or give its ASCII value. For example, the following two statements are equivalent

```
sString1 = Chr('A');
sString1 = Chr(65);
```

Clear a string using empty quotation marks:

```
sString1 = "";
```

String variables:

```
sOutgoing = sIncoming;
```

A string can be thought of as a table of characters. The number in square brackets is the character’s index. (Note that the index starts with the number zero.) The following code would result in sGreeting equaling “Hello!!!”

```
sGreeting = "Hello...";
sGreeting[5] = '!';
sGreeting[6] = '!';
sGreeting[7] = sGreeting[6];
```

A character element of a string variable may be treated like an Integer 32 value:

```
nNumber = sString2[1] * sString2[2];
```

The + operator is used to paste strings together. There is no limit to the number of + operators you can use on a line. The + operator must be used in an assignment statement:

```
sString1 = "Hello ";
sString2 = "world";
sString3 = "!";
After the three lines above, the following two lines would produce the
same result:
sString4 = sString1 + sString2 + sString3;
sString4 = sString1 + "world" + sString3;
```

Use the += operator to append one string to another and change the value of one of them into the result. In the following example, the value of sName would change to “Smith, John”:

```
sName = "Smith, ";
sFirstName = "John";
sName += sFirstName;
```

The Chr() keyword can be used to convert a numeric value into a one-element string:

```
sString5 = sString1 + sString2 + Chr('!');
sString5 = sString1 + sString2 + Chr(33);
```

Working with Pointers

Pointers can be tricky, but they are powerful tools. For more information on using pointers, see “Pointer Commands” on page 10-56.

For the following examples, assume that:

```
n1 = 5;
f1 = 9.2;
s1 = "test 123";
```

Set the pointer. The types must match or the control engine will generate an error.

```
pn1   = null;
pn1   = &n1;
pf1   = &f1;
ps1   = &s1;
pcht1 = &Powerup;
```

Use * to de-reference a pointer; it will then behave just like the variable to which it is pointing. The following two statements are equivalent:

```
n2 = *pn1 + *pf1
n2 = n1 + f1;
```

To see if a pointer is pointing to something, use the comparison operator == (see page 11-20) to compare it to null. This use is similar to standard ioControl condition commands such as Pointer Equal to NULL? For example:

```
n2 = pn1 == null;
n2 = null == pn1;
if (pt1[0] == null) then
```

To move a value from a pointer to another pointer:

```
pVariable0 = &*pVariable1;
```

To move a value from a pointer to a pointer table:

```
ptTable[0] = &*pVariable4;
```

Pointers are very useful when you don't know what variables need to be used until runtime. For instance, the next example uses a switch statement (see page 11-23) to determine which variable to use based on the day of the week. It then uses a pointer to perform a calculation using the correct variable.

```
switch (GetDayOfWeek())
   case 0:  // Sunday
      pn1 = n2;
      break
   case 6:  // Saturday
      pn1 = n3;
      break
   default: // Monday-Friday
      pn1 =5 n4;
      break
endswitch
```

Use the pointer to set the chosen variable.

```
*pn1 = n5 * f1 - 5;
```

Working with Tables

Following are some examples for using numeric, string, and pointer tables.

Numeric tables:

```
ntTable1[0] = 1;
ntTable1[1] = 2.0;
ntTable1[2] = nVar1;
ntTable1[3] = ntTable1[2];
ntTable1[4] = ntTable1[ntTable1[0]];
ntTable1[5] = nVar1 + ntTable1[2] * 3.1;
nVar1 = ntTable1[0];
nVar1 = (ntTable1[0] + ntTable1[1]) * ntTable1[2];
```

String tables:

```
stStrT1[0] = "Hello, ";
stStrT1[1] = "world";
stStrT1[2] = stStrT1[0] + " " + stStrT1[1] + Chr('!');
sString1 = stStrT1[2];
```

Pointer tables:

```
ptTable6[0] = &*pVariable2;
ptTable6[1] = nVar1;
ptTable6[2] = stStrT1[0];
stStrT1[0] = *ptTable6[2];
```

Pointer tables. Note that types are not checked when putting pointers into a pointer table. However, when a pointer is moved from a pointer table element into a pointer variable, the types are checked at runtime by the control engine and must match. For example, assume that the following elements have been placed in table ptPointT:

```
ptPointT[0] = null;
ptPointT[1] = &nLED_A;
ptPointT[2] = &fTemp;
ptPointT[3] = &sString1;
ptPointT[4] = &Powerup;
```

Based on this information, the first two of the following statements are good. The third one is bad and will cause a control engine error, because the element at ptPointT[3] is a string and therefore does not match the variable pntl, which is defined as an integer 32:

```
pn1 = ptPointT[1];
pf1 = ptPointT[2];
pn1 = ptPointT[3];
```

OptoScript Expressions and Operators

OptoScript includes mathematical expressions as well as comparison, logical, and bitwise operators. Because expressions and operators are built into the OptoScript language, several standard ioControl commands such as Multiply, Bit Shift, and Greater Than or Equal? are not used.

Using Mathematical Expressions

Addition

```
nCount = nLast_Count + 2;
fPressure = 1.5 + fReading;
nTotal = nMonday + nTuesday + 10;
```

Subtraction

```
nNumber_A = nNumber_B - 250;
fRange = fMax_Temp -
fMin_Temp;
```

Multiplication

```
nQuantity = nBoxes * 12;
nHours = nSeconds * 60 * 60;
fMax_Speed = fSpeed * 16.52;
```

Division

```
nBoxes = nCount / 6;
fConversion = fLimit / 2.0;
```

Modulo division. If any argument is a float, it is rounded to an integer before the division occurs.

```
nVar1 = nVar2 % 2;
nVar1 = 2 % nVar2 % nVar3;
fFloat1 = fFloat2 % 2.5;
```

Mixture of operators.

```
nAvg = (nHrs_A + nHrs_B) / 2;
nVar1 = fFloat2 + nVar3 * 4;
```

Use parentheses to clarify groupings and meaning. You can use an unlimited number of parentheses.

```
nVar1 = nVar2 * (fFloat2 - 2.0);
nVar1 = (nVar2 + 2) * (nVar3 + (fFloat1 / (fFloat2 - 2)) - 3);
```

The *, /, and % operators have greater precedence than + and -. (See page F-9 for the order of precedence.) In the following lines, line #1 is equivalent to line #3, not to #2.

```
n1 = n2 + n3 * n4;
n1 = (n2 + n3) * n4;
n1 = n2 + (n3 * n4);
```

Using Comparison Operators

All OptoScript comparison operators return an Integer 32 value of zero (false) or of non-zero (true). OptoScript supports the following comparison operators for comparing two numeric values:

Operator and Meaning		Example
==	equal	`nVar1 = nVar2 == fFloat3;`
<>	not equal	`nVar1 = nVar2 <> fFloat3;`
<	less than	`nVar1 = nVar2 < fFloat3;`
<=	less than or equal	`nVar1 = nVar2 <= fFloat3;`
>	greater than	`nVar1 = nVar2 > fFloat3;`
>=	greater than or equal	`nVar1 = nVar2 >= fFloat3;`

More complex examples:

```
nVar1 = (nVar2 * 2) == (fFloat3 / 9.5);
nVar1 = (nVar2 * 2) < (fFloat3 / 9.5);
```

You can also use a comparison operator to test whether two strings are equal. For example:

```
nVar1 = sString1 == sString2;
nVar1 = sString1 == "abc";
nVar1 = sString1 == stStrT1[0];
nVar1 = stStrT1[0] == stStrT1[1];
```

When you use a comparison operator in an *if* statement, it isn't necessary to put the result in a variable because the result is used (consumed) by the *if*:

```
if (fICTD_Input <= Avg_Temp) then
  Fan_A = 0;
endif
```

Using Logical Operators

All OptoScript logical operators return an Integer 32 value of zero (false) or of non-zero (true). OptoScript supports the following logical operators for numeric values:

Operator and Meaning		Example
`and`	Result is true if both values are true	`nVar1 = nVar2 and nVar3;`
`or`	Result is true if at least one value is true	`nVar1 = nVar2 or nVar3;`
`xor`	Result is true if only one value is true	`nVar1 = nVar2 xor nVar3;`
`not`	invert the logical value	`nVar1 = not nVar2;`

Any number of logical operators can be chained together:

```
nVar1 = nVar2 and nVar3 and nVar4;
nVar1 = nVar2 and nVar3 or nVar4;
```

The *not* operator precedes a value (it only takes a value on its right-hand side):

```
nVar1 = not nVar2;
```

Logical operators are left-associative. For example, these two lines are equivalent:

```
nVar1 = nVar2 and nVar3 or
nVar4;
nVar1 = (nVar2 and nVar3) or
nVar4;
```

The following two lines are equivalent:

```
nVar1 = not nVar1 and not
nVar2;
nVar1 = (not nVar1) and (not
nVar2);
```

Logical operators can be combined with comparison operators to create complex logical expressions:

```
nVar1 = (nVar2 < 1) and (nVar3 == 6.5);
nVar1 = (nVar2 < 1) and (sString1 == "abc");
nVar1 = ((nVar2 < 1) and (nVar4 xor nVar5) or (not (fFloat1 ==
fFloat2));
nVar1 = not (nVar2 < 5); // same as  "nVar1 = nVar2 >= 5;"
```

When you use a logical operator in an *if* statement, it isn't necessary to put the result in a variable because the result is used (consumed) by the *if*:

```
if (Motor_1 or Motor_2) then
  Motor_3 = 0;
endif
```

Using Bitwise Operators

All OptoScript bitwise operators operate on integer values. OptoScript supports the following bitwise operators:

`bitand`	(bitwise and)
`bitor`	(bitwise or)
`bitxor`	(bitwise xor)
`bitnot`	(bitwise not)
`<<`	(left shift)
`>>`	(right shift)

Use the *bitwise and* operator to *and* together the two values bit by bit:

```
n1 = n2 bitand 2;
n1 = n2 bitand n3;
```

Hex literals can be convenient:

```
n1 = n2 bitand 0x0002;
```

Use the *bitwise or* operator to *or* together the two values bit by bit:

```
n1 = n2 bitor 2;
n1 = n2 bitor 0x0002;
n1 = n2 bitor n3;
```

Use the *bitwise xor* operator to *xor* together the two values bit by bit:

```
n1 = n2 bitxor 2;
n1 = n2 bitxor 0x0002;
n1 = n2 bitxor n3;
```

The *left-shift* operator shifts the left value's bits to the left by the right value:

```
n1 = n2 << 2;     // left shift n2's value by 2
n1 = n2 << n3;    // left shift n2's value by n3
```

The *right-shift* operator shifts the left value's bits to the right by the right value:

```
n1 = n2 >> 2;     // right shift n2's value by 2
n1 = n2 >> n3;    // right shift n2's value by n3
```

Precedence

For a list of operators from highest to lowest precedence, see "Operators" on page F-9.

OptoScript Control Structures

OptoScript provides the following structures to control the flow of logic in the code:

- "If Statements" (below)
- "Switch or Case Statements" on page 11-23
- "While Loops" on page 11-23
- "For Loops" on page 11-24
- "Repeat Loops" on page 11-24

If Statements

If statements offer branching in logic: if statement A is true, then one action is taken; if statement A is false (or statement B is true), a different action is taken. *If* statements are very flexible; here are several examples of ways you can use them.

Any numeric value can be tested by the *if* statement:

```
if (n1) then
  f1 = 2.0;
endif
```

Since a comparison operator returns an Integer 32 value, it can be used as the test value:

```
if (n1 > 3) then
  f1 = 2.0;
  f2 = 6.5;
endif
```

Complex logical operations can also be used:

```
if ((n1 > 3) and (not n1 == 6)) then
  f1 = 2.0;
  f2 = 6.5;
endif
```

An optional `else` statement can be added:

```
if (n1 > 3) then
  f1 = 2.0;
  f2 = 6.5;
else
  f3 = 8.8;
endif
```

Multiple `elseif` statements can be used to chain together several tests. The `else` statement is still allowed at the end.

```
if (n1 > 3) then
  f1 = 2.0;
  f2 = 6.5;
elseif (n1 < -3) then
  f3 = 8.8;
elseif (n1 == 0) then
  f3 = f1 * f2;
else
  f1 = 0;
  f2 = 0;
  f3 = 0;
endif
```

If statements can be nested. Each `if` requires an `endif`:

```
if (n1 > 3) then
  f1 = 2.0;
  f2 = 6.5;

  if (n1 % 10) then
    f1 = f1 * 2;
    f2 = f2 * 3;
  else
    f3 = 0;
  endif
endif
```

Switch or Case Statements

A *switch* or *case* statement also offers branching logic and can be used in place of *if* statements when the expression can match one of a number of numeric values. The value for each case can be a numeric constant or a mathematical expression only. Comparisons and logical operators cannot be used in cases, nor can strings. If a case involves a float, the float is converted to an integer before use. Notice that only one case can be tested at a time.

Here's an example of a switch statement.

```
switch (nNumber)
  case 1:
    f1 = 10;
    break
  case 2:
    f1 = 15;
    break
  case (n2 * 2):
    f1 = 20;
    break
  default:
    f1 = 0;
    f2 = -1;
    break
endswitch
```

The value of the expression in parentheses, nNumber, is compared to each of the cases. If the case matches the value of nNumber, the action is taken.

Make sure you use a colon (`:`) after each case.

If a case matches the value of nNumber, the `break` statement after the action immediately exits the switch. Notice that a semicolon is not used after `break`.

You can use a mathematical expression as a case.

If no case matches, the `default` action is taken. Using a default is optional; if you use it, it must be at the end of the list.

A `switch` statement must be followed by `endswitch`.

While Loops

The *while* loop is used to execute a list of statements while a given condition is true. The condition is tested at the beginning of each loop.

For example, this loop sets the first five elements (elements 0 through 4) of a table (ntTable) to a value of 10:

```
nIndex = 0;
while (nIndex < 5)
  ntTable[nIndex] = 10;
  nIndex = nIndex + 1;
wend
```

Initialize the counter.

Execute loop if condition is true.

Set the table element.

Increment the counter.

While loops can be nested and can contain other kinds of program statements. Each `while` needs a matching `wend` at the end. For example:

```
n1 = 0;
 while (n1 < 100)
  while ((n1 > 50) and (n1 < 60))
     nt1[n1] = n1 * 100;
     n1 = n1 + 1;
   wend
   nt1[n1] = n1;
   n1 = n1 + 1;
 wend
```

Repeat Loops

Repeat loops, in contrast to *while* loops, are used to execute a list of statements until a given condition is true. Because the condition is tested at the end of each loop, the content of the loop will always be executed at least once.

This example sets the first five elements of ntTable to 10. Compare this example to the example for *while* loops to see the difference.

```
nIndex = 0;
repeat
 ntTable[nIndex] = 10;
  nIndex = nIndex + 1;
until (nIndex >= 5);
```

Initialize the counter.

Set the table element.

Increment the counter.

Execute loop until condition is true.

Repeat loops can be nested and can contain other kinds of program statements. Each `repeat` statement needs a matching `until` statement at the end.

For Loops

For loops can be used to execute a list of statements a certain number of times.

The `for` line sets up a predefined initial value and a predefined final value for the counter that counts the repetitions. The line also includes the steps by which the counter gets from its initial value to its final value (step 1 counts by ones; step 2 counts by twos, and so on). The `step` is required. The counter can be any numeric variable or I/O point, but its value will always be a whole number. The initial value, final value, and step can be any numeric expression; they are converted to integer 32s.

CAUTION: *A step value of zero creates an infinite loop. A float step value between –0.5 and 0.5 also creates an infinite loop, since it is rounded to zero when converted to an integer 32.*

This example results in nVariable equaling 6:

```
nVariable = 1;
 for nCounter = 0 to 4 step 1
   nVariable = nVariable + 1;
 next
```

The counter starts at zero, and its final value is 4. It will count up one step at a time.

The `for` loop must end with `next`.

The *for* loop counter can be used in the loop. This example sets the first five elements of table ntTable to 10:

```
  for nIndex = 0 to 4 step 1
    ntTable[nIndex] = 10;
  next
```

Other step amounts can be used, including negative steps. Do not use a zero step, which creates an infinite loop. This example sets elements 0, 2, and 4 of ntTable to 20:

```
  for nIndex = 0 to 4 step 2
    ntTable[nIndex] = 20;
  next
```

Predefined values can be a numeric expression, but they are evaluated only at the beginning of the loop. For instance, the following example will loop 0 to 15 because the upper limit of nSide*3 is evaluated only at the beginning of the loop, not each time through the loop:

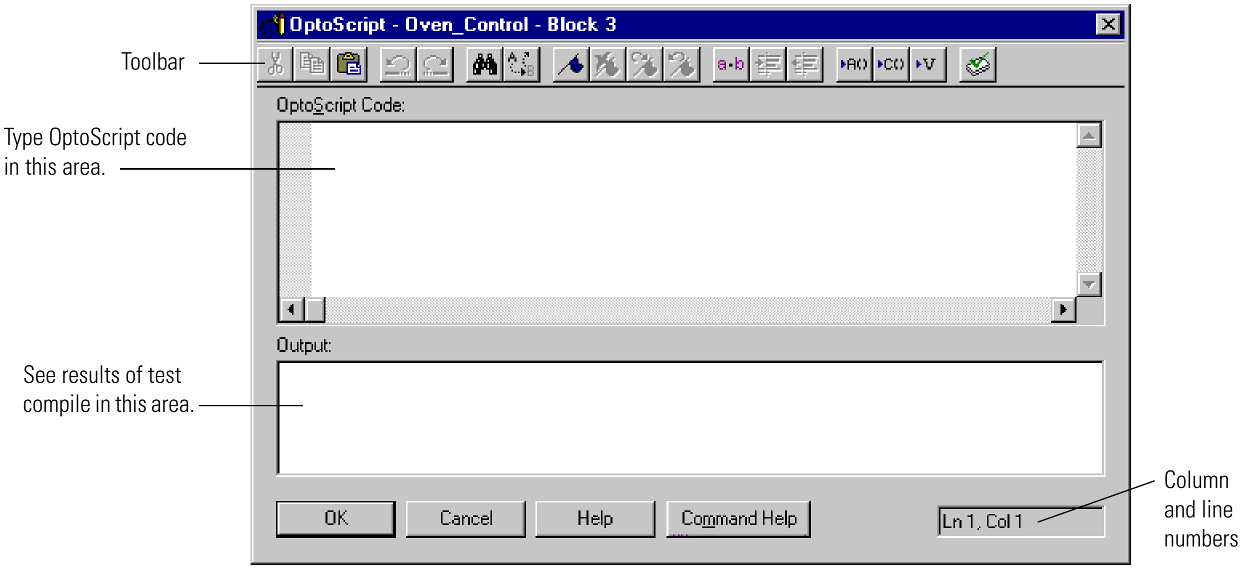
```
  nSide = 5;
  for nLength = 0 to (nSide * 3) step 1
    nSide = 1;
  next
```

For loops can be nested and can contain other types of statements. Each `for` requires a `next` at the end.

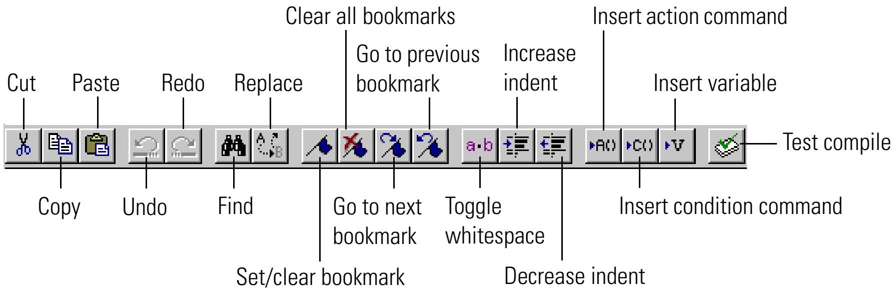
Using the OptoScript Editor

1. To use the editor, create an OptoScript Block in the flowchart where you want the code to appear. (For more information on creating charts and blocks, see Chapter 8, "Working with Flowcharts.") Double-click the OptoScript block to open the editor.

The editor is similar to the editor for Microsoft Visual Basic®. You can resize the editor window as needed to see the code.


Toolbar
Type OptoScript code in this area.
See results of test compile in this area.
Column and line numbers
OptoScript - Oven_Control - Block 3
OptoScript Code:
Output:
OK
Cancel
Help
Command Help
Ln 1, Col 1

The toolbar includes the following buttons:



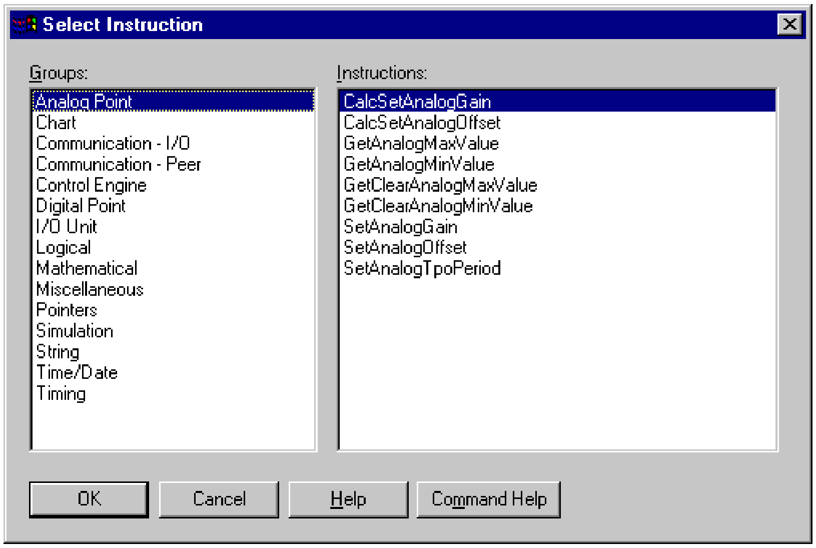
2. Begin typing OptoScript code in the top area.

You'll notice that what you type is automatically color-coded to help you:

- Blue—operators and control structures
- Purple—values
- Green—comments
- Black—commands and names of variables, I/O points, charts, and other items
- Red—string literals.

If you want to see white-space marks to help line up code, click the Toggle Whitespace button in the toolbar. To hide the marks, click the button again.

3. To use a command, place your cursor in the OptoScript code where you want the command to appear. Click the Insert Action Command ▸A() or Insert Condition Command ▸C() button in the toolbar.



4. In the Select Instruction dialog box, select the command group from the left-hand column, and then select the command name from the right-hand column.

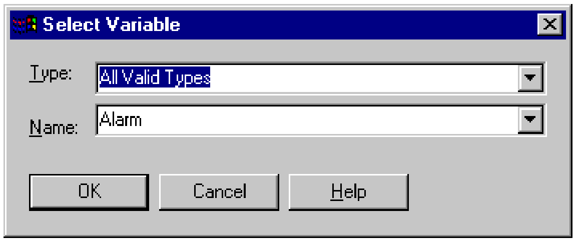
   For information on any command, highlight it and click Command Help, or just double-click the command name.

   NOTE: If you know the command name, you can just type it into the OptoScript code. Remember that OptoScript command names may be different from standard ioControl commands. See Appendix E, "OptoScript Command Equivalents" for more information.

5. Click OK.

   The command appears in the OptoScript code.

6. To use a variable, table, I/O unit or point, chart, counter, timer, or similar item, place your cursor where you want the item to appear in the code. If you know the item's exact name, enter it and skip to step 8. If you're not sure of the item's name, click the Insert Variable button ▸V in the toolbar.



Type: All Valid Types

Name: Alarm

7. From the Type drop-down list in the Select Variable dialog box, choose the type of item you want to use. From the Name drop-down list, choose the item. Click OK.

The item appears in the code.

8. Use the TAB key on the keyboard as you type to indent lines as needed. To increase or decrease indentation for a line of code you've already typed, highlight the line and click the Increase Indent or Decrease Indent button in the toolbar.

9. Enter comments to document what the code does, so anyone who must debug or maintain the code can clearly see your intentions.

   Comments appear in green. Line comments must be preceded by two slashes, for example:

   ```
   // This is a line comment.
   ```

   Block comments must be preceded by one slash and an asterisk, and be followed by the same two elements in reverse. For example:

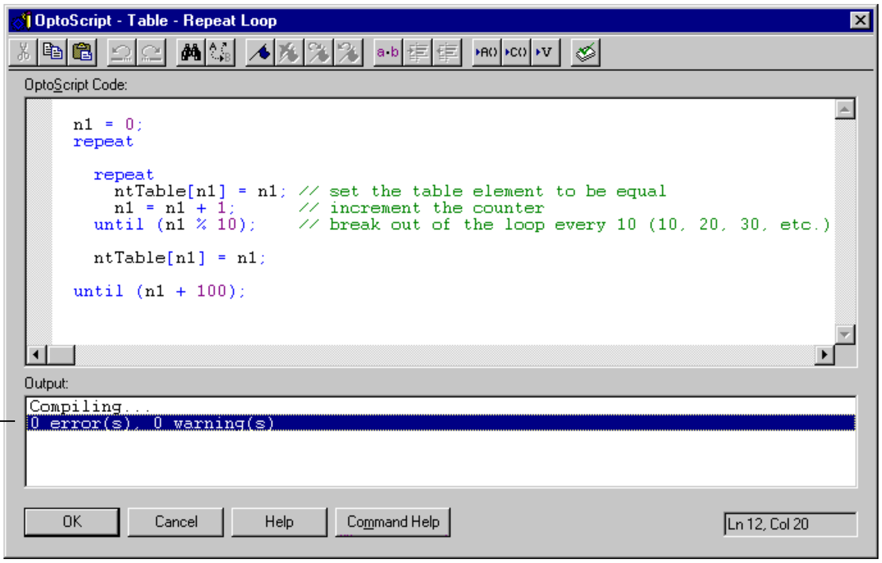
   ```
   /* This is a block comment that goes
         beyond one line. */
   ```

10. Use the Bookmark buttons in the toolbar as needed to set or clear temporary bookmarks within the code and to move between them.

    Bookmarks mark lines of code so you can easily find them and jump from one bookmark to the next. Bookmarks remain only while the editor is open; they are not saved when the dialog box is closed.

11. When you have finished entering all the code for an OptoScript block, click the Test Compile button in the toolbar to compile the code for this block.

    The code is compiled, and the results appear in the bottom part of the OptoScript window:



```
n1 = 0;
repeat

  repeat
    ntTable[n1] = n1; // set the table element to be equal
    n1 = n1 + 1;      // increment the counter
  until (n1 % 10);    // break out of the loop every 10 (10, 20, 30, etc.)

  ntTable[n1] = n1;

until (n1 + 100);
```

```
Compiling...
0 error(s), 0 warning(s)
```

Results after code is compiled

NOTE: The next time the chart is compiled, all OptoScript code within the chart will be compiled again.

If errors are found, you can fix them now or later. Begin with the first one (the one on the lowest-numbered line), since later errors are often a result of earlier errors. To check a command, place the cursor anywhere in the command name and click the Command Help button. If you need to add variables or other items that don't exist in the strategy, do so after step 12.

12. When you have finished with the code in this OptoScript block, click OK to save your work and close the editor.

You return to the flowchart.

Troubleshooting "Unable To Find" Errors

See also, "Troubleshooting Syntax Errors" below.

If you test compile an OptoScript Block and receive "unable to find" errors, try the following suggestions.

For Commands

Check the exact spelling of the command, including upper and lower case. OptoScript commands are similar to standard ioControl commands, but contain no spaces and some abbreviations.

Also check that the command is necessary in OptoScript. Some common commands, including comparison commands such as Less? and mathematical commands such as Add, are replaced with operators built into the OptoScript language. Check Appendix E, "OptoScript Command Equivalents" for equivalent OptoScript commands.

The easiest way to make sure the command you enter is valid is to enter it by clicking one of the Insert Command buttons in the OptoScript Editor and choosing the command from the Select Instruction dialog box.

For Variables or Other Configured Items

Variables, I/O units and points, counters, and other configured items in your strategy—as well as charts—have usually been created before you use them in OptoScript code. Check their exact spelling, including underscores and upper and lower case, to make sure they are correct in the code. The easiest way to make sure spelling is correct is to enter the variable or other item by clicking the Insert Variable button in the OptoScript Editor and choosing the item from the drop-down lists.

If the item has not yet been configured or created, use the normal ioControl methods to do so. For help, see the topicschapters in this guide on configuring I/O and using variables.

Troubleshooting Syntax Errors

Check for the following common syntax errors:

Missing Code

Check for obvious errors first. For example, make sure nothing essential has been left out of (or unnecessarily added to) a statement:

Sample Statement	Should Be	Missing Code
`iTotal = x + y + ;`	`iTotal = x + y + z;`	Last operator missing a variable
`iTotal = x + y + z`	`iTotal = x + y + z;`	Semicolon missing
`sGreeting = Hello!"`	`sGreeting = "Hello!"`	First quotation mark missing on the string
`iTime = Get Hours;`	`iTime = GetHours();`	Extra space in command name; parentheses missing after the command
`x = (1 + (x - y);`	`x = (1 + (x - y));`	Parentheses mismatched (last half missing)

Check to make sure operators are used correctly. You may want to review "OptoScript Expressions and Operators" on page 11-19.

If you are using control structures such as loops or *if* statements, especially if they are nested, make sure all required elements are present. For example, every `if` must have a `then` and an `endif`. See "OptoScript Control Structures" on page 11-22 for more information.

Type Conflicts

Type conflicts are caused when different data types are incorrectly mixed. For example, you cannot assign an integer to a string. Make sure data types are correct. It is easier to keep track of data types if you use Hungarian notation when naming variables. See "Variable Name Conventions" on page 11-14 for help.

Debugging Strategies with OptoScript

Before trying to debug strategies containing OptoScript code, make sure the code has been compiled within each block, or choose Compile All to do all blocks at once.

When you begin debugging the strategy, start by stepping through whole blocks. If you run across a problem, then step within that block. Stepping within the block is discussed in "Choosing Debug Level" on page 7-13.

CHAPTER 12

Using Subroutines

Introduction

This chapter shows you how to create and use subroutines.

In this Chapter

About Subroutines

A subroutine is a custom command that represents a series of commands. Subroutines are useful anytime you have a group of commands that is repeated in a Strategy or used in more than one strategy. Subroutines are built using the same tools and logic used to create charts. Once built, you can call them at any time from any chart in any strategy. (You cannot call a subroutine from another subroutine, however.)

Like charts, subroutines start at one block and proceed sequentially through command blocks to the end. They use variables, inputs, and outputs. They can use OptoScript code. Each subroutine is displayed in its own window, and you can open and view several subroutine windows at the same time.

Unlike charts, however, subroutines are independent from a strategy. You don't need to have a strategy open to create or change a subroutine. And if you do have a strategy open, creating a subroutine has no effect on the open strategy unless you specifically link them together. (Debugging a subroutine, however, requires that it be called from a strategy.)

A second important difference between subroutines and charts is that subroutines offer two ways to work with variables and other logical elements: they can be passed in or they can be local to the subroutine.

- **Passed-in items** are passed into the subroutine by the strategy. They are referenced when the subroutine is executed, and (if they are variables) they are permanently affected by the subroutine. For example, you could create a subroutine to add 3.0 to a passed-in float variable. When the subroutine ends, the float variable will contain a new value. Passed-in items are called *subroutine parameters*, and you can use up to 12 of them in a subroutine.

- **Local items** are created when a subroutine begins, and they are destroyed when it ends. For example, a subroutine could take a passed-in item, copy it to a local variable, and add 3.0 to that local variable for use within the subroutine. The local variable is created when the subroutine is called, and it disappears when the subroutine ends.

Data Types for Subroutines

The following data types may be used in subroutines for both passed-in items and local items:

- Numeric variables (integers, floats, and timers)
- Numeric literals (integers and floats). Other types can be passed into the subroutine through literals, however; see below for more information.
- Numeric tables
- String variables
- String literals
- String tables
- Communication handles

In ioControl Professional, the following data types are also supported:

- For passed-in items: I/O points, I/O units, and pointer tables
- For local items: pointer variables

The following data types are not supported in subroutines: PID loops, event/reactions, and charts.

Although most variables passed in and out of a subroutine must be of a specific type, *literals* that are passed *into* subroutines can take several types. Using a string literal, you can pass in either a string literal or a string variable. Using a numeric literal, you can pass in an Analog Point, a Digital Point, an integer variable, a float variable, or a timer variable. If you are familiar with other programming languages, literals are similar to “passed by value” parameters, while variables are like “passed by reference” parameters.

This flexibility in using literals makes it easier to use a subroutine in multiple strategies. For example, a literal passed into a subroutine from two strategies might be a float value in one strategy and an analog point in the other.

Creating Subroutines

Tips for Subroutines

As a general rule, keep subroutines as small as possible to do the job they're intended for. Extra variables and unnecessarily large table sizes can affect the memory available for running subroutines.

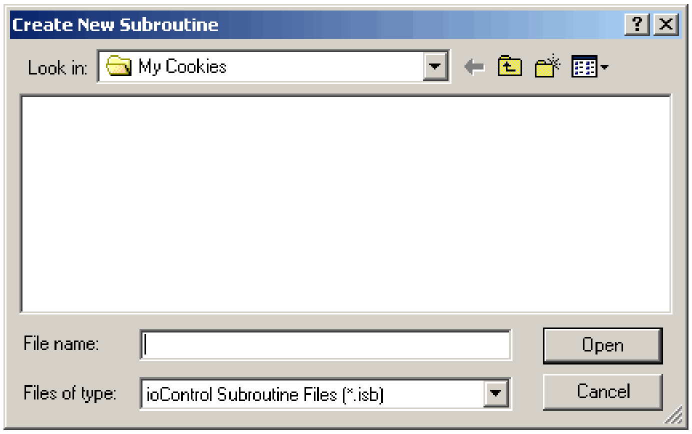
A Put Status In parameter appears automatically in every subroutine. This parameter is used to let you know whether the subroutine was called successfully, in the same way that function commands return a status. Make sure that you always check the status code after calling a subroutine. Subroutine status codes are:

0	Success
-67	Out of memory
-69	Null object error. Make sure you are not passing a pointer that points to null.
-72	Nesting too deep (future use)

Drawing the Flowchart

1. In the ioControl main window (with or without a strategy open), choose Subroutine➞New.

   The Create New Subroutine dialog box appears.



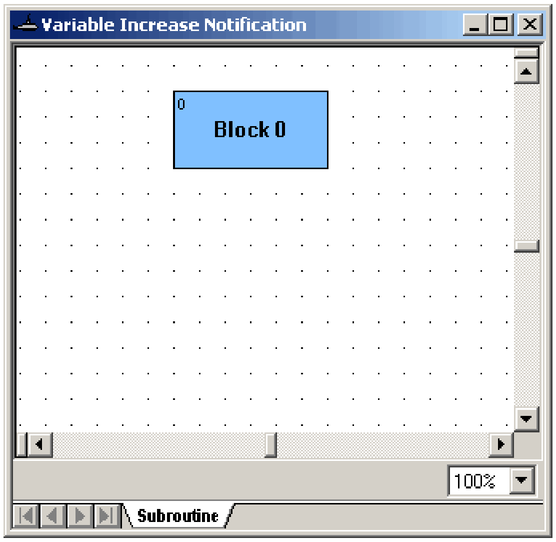
2. Enter a subroutine name.

   The subroutine name will become a command (instruction) in ioControl. It's a good idea to make it a descriptive name indicating the purpose of the subroutine, for example, "Variable Increase Notification." You cannot use the name of any existing command (for example, "Add").

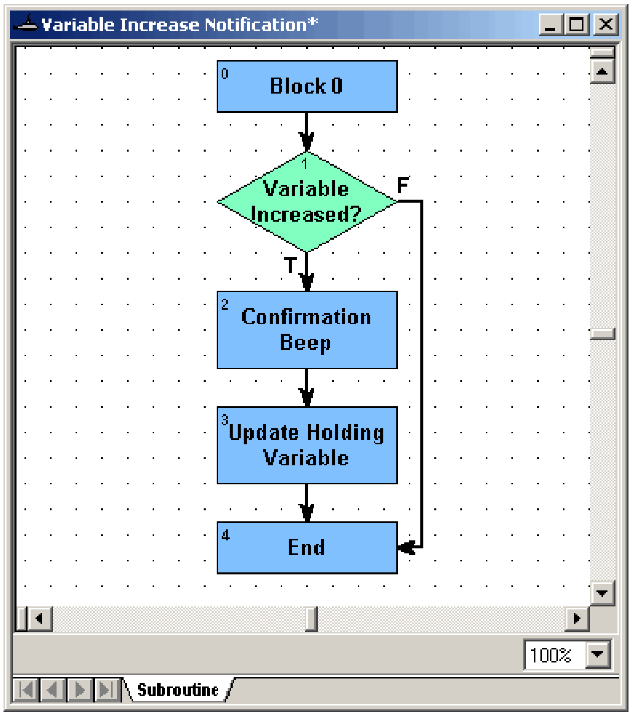
3. Navigate to the directory where you want to store the subroutine and click Open.

Unlike strategies, multiple subroutines can be saved to the same directory.

A new subroutine window is created.



4. Add blocks and connections and name the blocks as you would in a chart, as shown in the example below.



You can also copy existing flowchart blocks from another subroutine or chart and paste them into the new subroutine. See "Cutting, Copying, and Pasting Elements" on page 8-12.

5. Save the subroutine by selecting Subroutine→Save.

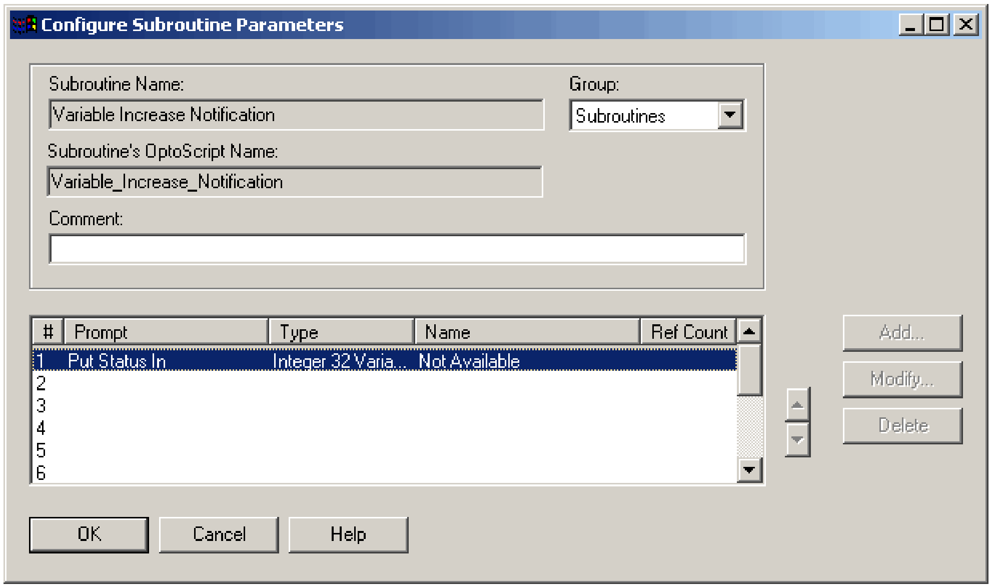
Configuring Subroutine Parameters

Before you can call a subroutine from a Strategy, you must configure the variables and other logical items that are passed into it. These passed-in items, called subroutine parameters, are the only information that is shared between a subroutine and the calling strategy. Twelve parameters can be passed into a subroutine, and since a table can be a parameter, those 12 parameters can include a large amount of data.

An item passed into a subroutine may be called by one name in the strategy and by another name in the subroutine. In fact, if a subroutine is used for more than one strategy, it is good practice to select generic names in the subroutine. For example, if you create a subroutine to average values in any float table, the table might be named Float_Table in the subroutine. You could use this subroutine to average pressures in a table named Pressure_Values from a strategy, and Pressure_Values would be referred to as Float_Table in the subroutine.

1. With the subroutine open, select Subroutine→Configure Parameters.

   The Configure Subroutine Parameters dialog box appears.



Subroutine Name: Variable Increase Notification

Group: Subroutines

Subroutine's OptoScript Name: Variable_Increase_Notification

Comment:

#	Prompt	Type	Name	Ref Count
1	Put Status In	Integer 32 Varia...	Not Available	
2				
3				
4				
5				
6				

   In this dialog box you determine the way the subroutine is called from the strategy.

2. From the Group drop-down list, choose the command group you want the subroutine to appear in.

   For example, If you create a subroutine to find the average value of several variables, for example, you could choose Mathematical as the command group. The default group is Subroutines.

3. (Optional) Enter a comment to explain the purpose of the subroutine.

4. Notice that one parameter, Put Status In, has been automatically entered for you.

This parameter is used to return status information on the subroutine, and it always appears at the bottom of the parameter list. A subroutine essentially becomes a command within a chart; subroutines are similar to function commands that return a status. Since the system itself returns this status parameter, the name of the status parameter is not available in the subroutine. When you add the subroutine to a strategy, you choose the variable the status will be placed in.

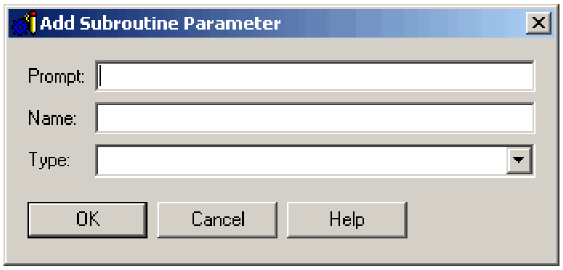
IMPORTANT: *Make sure you always check returned status codes for all subroutines. Subroutine status codes and their meanings are:*

0	Success
-67	Out of memory
-69	Null object error. Make sure you are not passing a pointer that points to null.
-72	Nesting too deep (future use)

5. For each parameter you add, do the following steps.

   NOTE: What you enter here appears in the Add Instruction dialog box when the subroutine is called from within the strategy. See page 12-8 for an example.

   a. Highlight the first empty line (below the Put Status In parameter) and click Add to open the Add Subroutine Parameter dialog box.



   b. In the Prompt field, enter the prompt text you want to show in the Add Instruction dialog box in the strategy.

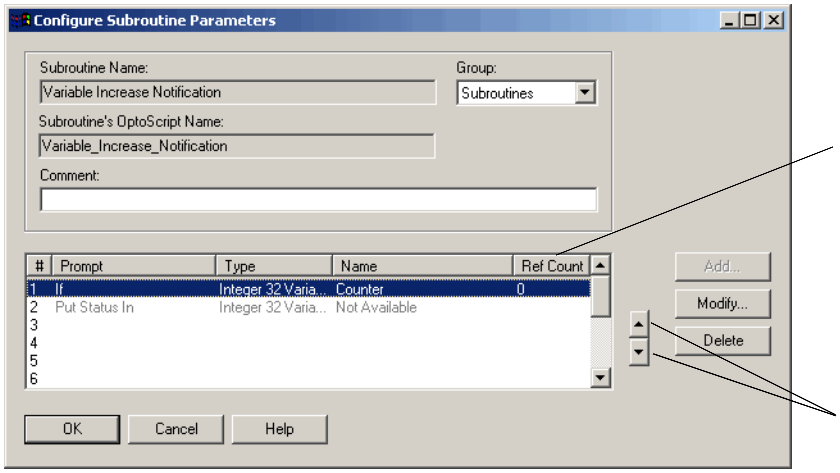
   c. In the Name field, enter the name of the parameter (the argument) as it will be referred to in the subroutine. This name is used within the subroutine only.

   d. From the Type drop-down list, choose the type of item to be passed into the subroutine.

   Use variables for values the subroutine changes (passed by reference); use literals for values the subroutine uses but does not change (read-only or passed by value).

   e. Click OK.

The parameter appears in the Configure Subroutine Parameters dialog box, above the Put Status In parameter.



Configure Subroutine Parameters

Subroutine Name: Variable Increase Notification

Group: Subroutines

Subroutine's OptoScript Name: Variable_Increase_Notification

Comment:

#	Prompt	Type	Name	Ref Count
1	If	Integer 32 Varia...	Counter	0
2	Put Status In	Integer 32 Varia...	Not Available	
3				
4				
5				
6				

Reference count

Up- and down-arrow buttons

6. Repeat step 5 for each parameter. To change a parameter, highlight it and click Modify. To change the order of a parameter in the list, highlight it and click the up- or down-arrow button in the dialog box. To delete a parameter, highlight it and click Delete.

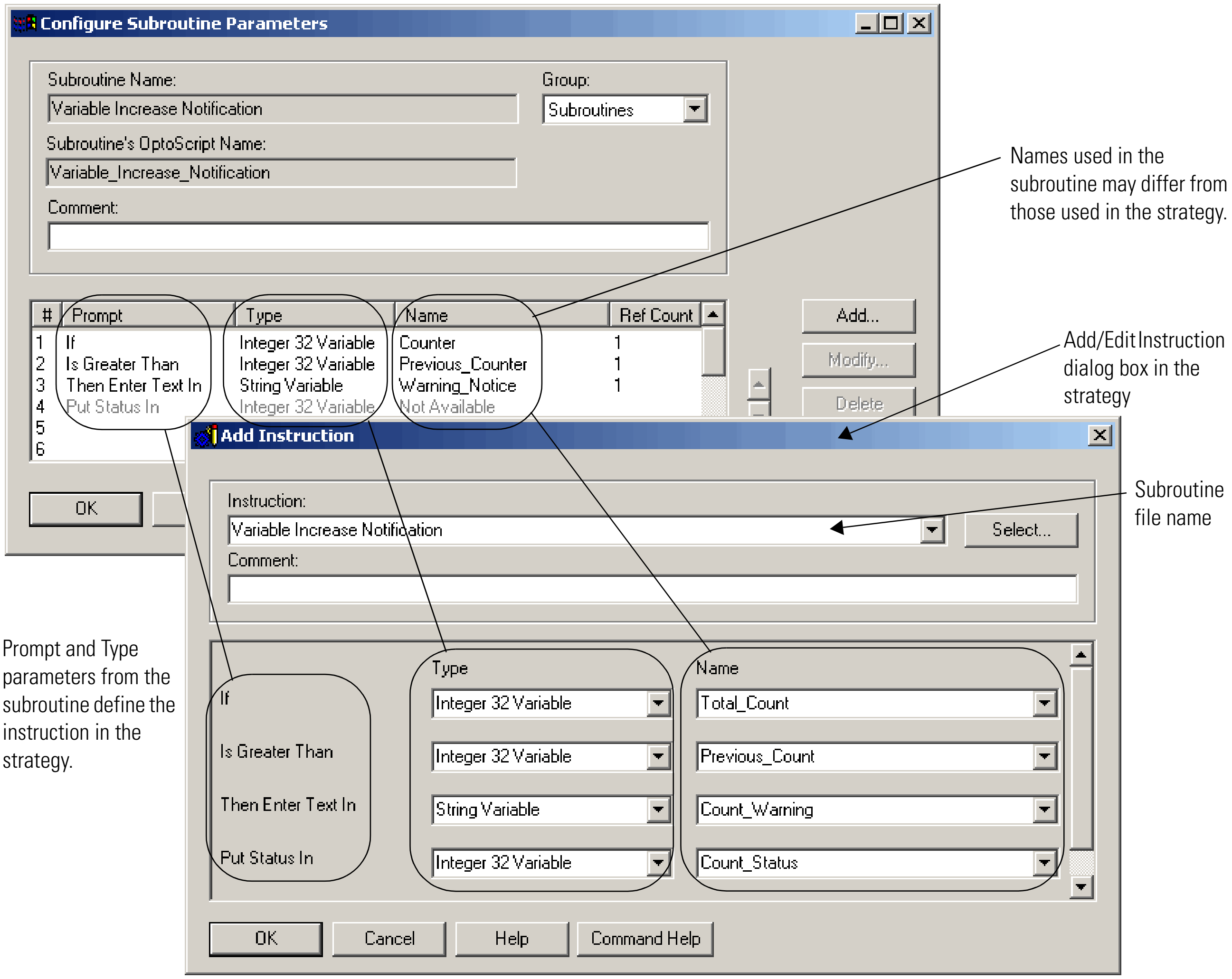
   NOTE: You cannot delete the Put Status In parameter or change its order in the list, and you cannot delete a parameter that has a reference count greater than zero (indicating that it is used in the subroutine). Also, if you add or delete parameters after including a subroutine in a strategy, you may receive an error and will need to add the subroutine to the strategy again.

7. When the parameters appear the way you want them in the list, click OK.

   The parameters you have named can now be used in the subroutine's commands.

Configured Parameters Example

Here's an example of a completed Configure Subroutine Parameters dialog box, showing three parameters to be passed into the subroutine. When the subroutine is called from the strategy, these parameters appear in the Add Instruction dialog box:



Configure Subroutine Parameters

Subroutine Name: Variable Increase Notification

Group: Subroutines

Subroutine's OptoScript Name: Variable_Increase_Notification

Comment:

#	Prompt	Type	Name	Ref Count
1	If	Integer 32 Variable	Counter	1
2	Is Greater Than	Integer 32 Variable	Previous_Counter	1
3	Then Enter Text In	String Variable	Warning_Notice	1
4	Put Status In	Integer 32 Variable	Not Available	
5				
6				

Add Instruction

Instruction: Variable Increase Notification

Comment:

	Type	Name
If	Integer 32 Variable	Total_Count
Is Greater Than	Integer 32 Variable	Previous_Count
Then Enter Text In	String Variable	Count_Warning
Put Status In	Integer 32 Variable	Count_Status

Names used in the subroutine may differ from those used in the strategy.

Add/Edit Instruction dialog box in the strategy

Subroutine file name

Prompt and Type parameters from the subroutine define the instruction in the strategy.

Adding Commands and Local Variables

Adding commands (instructions) to subroutines is exactly like adding instructions to charts. For help, see "Adding Commands" on page 9-18. If you are using OptoScript code within a subroutine, see Chapter 11 for help on creating code. You can copy and paste instructions from one block to another within the same subroutine; if you have trouble copying and pasting instructions from one chart or subroutine into another, simply paste the entire block and then modify it.

You may also need to add local items to be used in the subroutine only and discarded when the subroutine is finished. Adding variables to subroutines is also like adding variables to charts. For help, see "Adding Variables" on page 9-5.

Remember that the subroutine is separate from any strategy and can be called by any strategy. I/O units and points are specific to a strategy, so they cannot be added to a subroutine. Also, note that commands and OptoScript code within a subroutine can use only the passed-in and local items in the subroutine, not items in the strategy that calls the subroutine.

Compiling and Saving the Subroutine

1. With the subroutine open, select Compile→Compile Subroutine.

   When the subroutine has finished compiling, the cursor returns to its normal form.

2. Select Subroutine→Save.

Using Subroutines

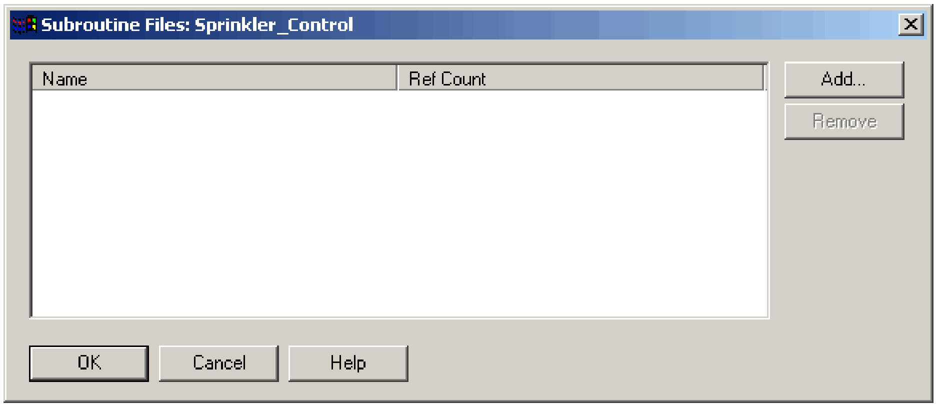
To use a subroutine in a Strategy, you include it in the strategy and then add it as a command (instruction) so it can be called from a chart.

Including a Subroutine in a Strategy

Since subroutines are independent of strategies, you must include the subroutine in the strategy before you can use it.

1. With the strategy open in Configure mode, double-click the Subroutines Included folder on the Strategy Tree (or click the Include Subroutines button on the toolbar, or select Configure→Subroutine Included).

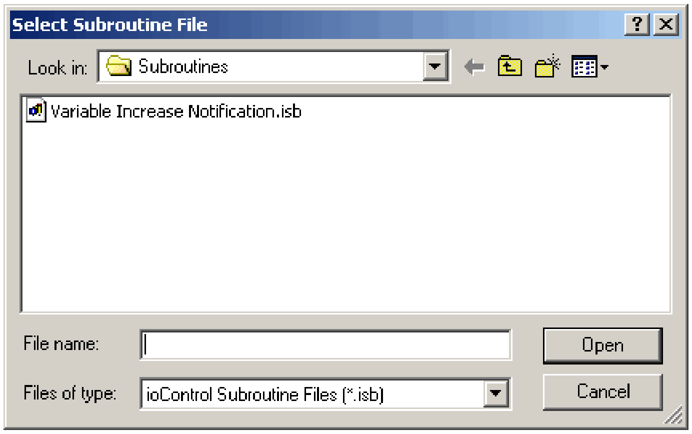
   The Subroutine Files dialog box appears, listing all subroutines currently included in the strategy. The example below shows no subroutines currently included.



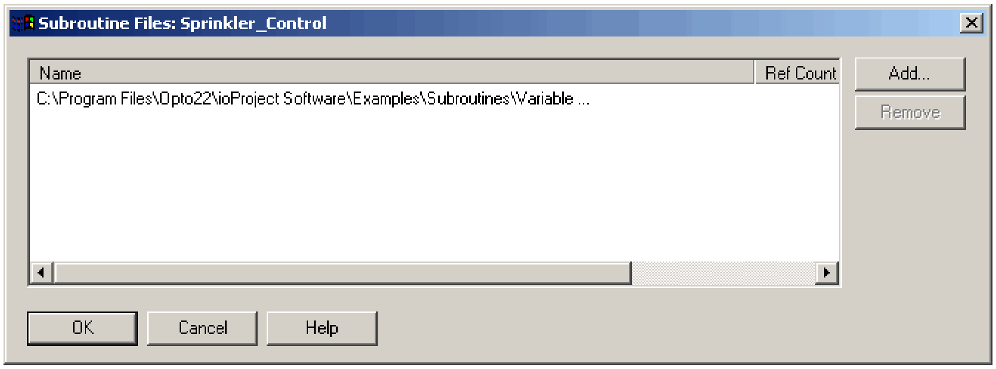
Subroutine Files: Sprinkler_Control

Name	Ref Count

2. Click Add.



3. Navigate to the directory containing the subroutine you want to add and double-click the subroutine.


Subroutine Files: Sprinkler_Control
Name
Ref Count
C:\Program Files\Opto22\ioProject Software\Examples\Subroutines\Variable ...
Add...
Remove
OK
Cancel
Help

4. When the full path to the subroutine appears in the Subroutine Files dialog box, click OK.

   The new subroutine appears in the Strategy Tree in the Subroutines Included folder.

Adding a Subroutine Instruction

You use a subroutine just like an ioControl command: by adding the subroutine instruction to a block in the chart.

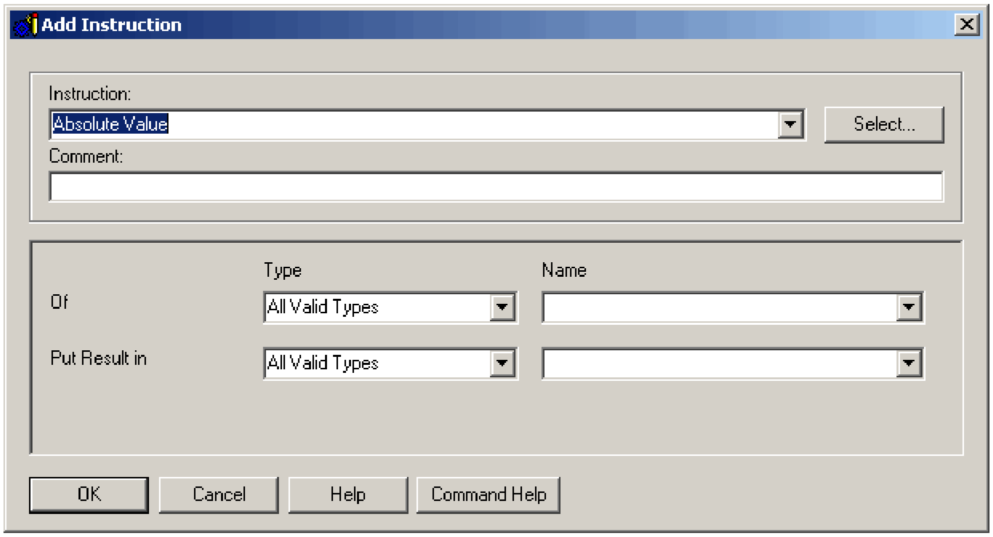
1. With the strategy open in Configure mode, open the chart that will use the subroutine.

2. Double-click the block that will call the subroutine.

   If it is an OptoScript Block, see "Using the OptoScript Editor" on page 11-25 for how to enter a command (instruction). For action or condition blocks, continue with step 3.

3. In the Instructions dialog box, click where you want the instruction to be placed, and then click Add.

The Add Instruction dialog box appears.



Add Instruction

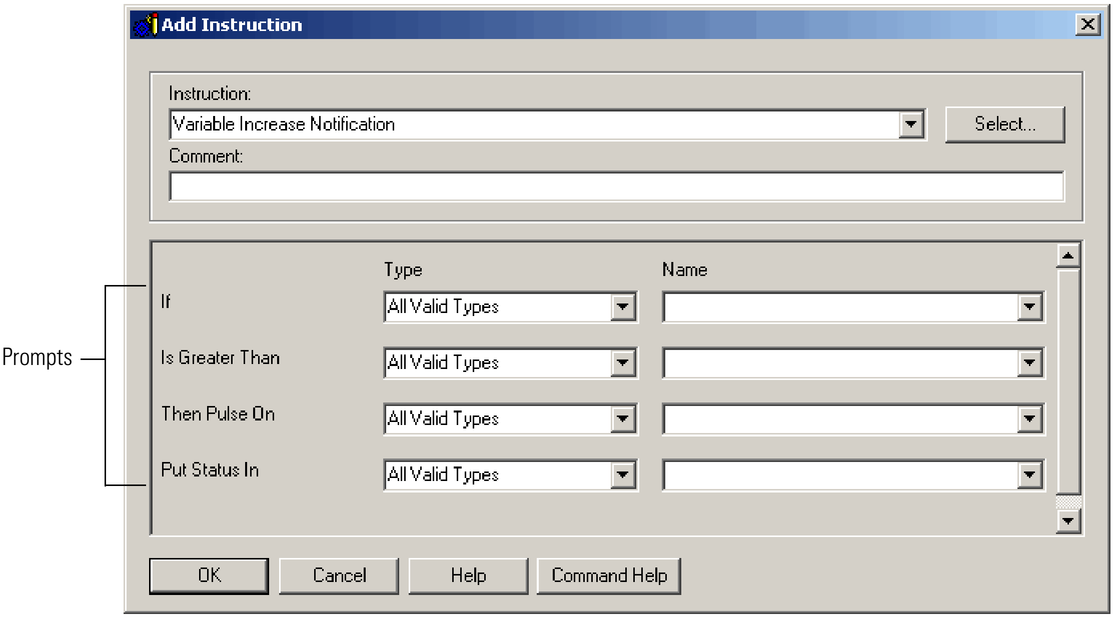
Instruction: Absolute Value

Comment:

	Type	Name
Of	All Valid Types	
Put Result in	All Valid Types	

4. In the highlighted Instruction field, enter the subroutine name (or choose it using the drop-down list, or click the Select button and locate it in the command group you chose).

   The subroutine command appears in the dialog box, just as any command would, and the prompts you entered when you configured the parameters appear also.



Add Instruction

Instruction: Variable Increase Notification

Comment:

	Type	Name
If	All Valid Types	
Is Greater Than	All Valid Types	
Then Pulse On	All Valid Types	
Put Status In	All Valid Types	

5. Choose the Type and Name for each prompt from the drop-down lists.

   You can configure variables on the fly as you would with any command. Remember that the Type was chosen when the parameters for the command were configured, so your Type choices may be limited.

6. When the Add Instruction dialog box is completed, click OK.

7. Click Close to close the Instructions dialog box and return to the chart.

   The chart is now set up to call the subroutine.

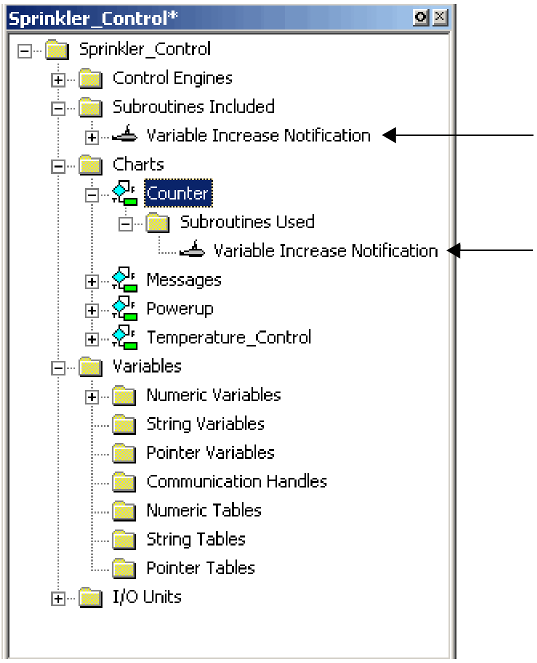
Debugging Subroutines

Debugging a subroutine is just like debugging a flowchart. When you are debugging a strategy that calls the subroutine, make sure the debug level is Full Debug (see page 7-13). Then use the Step Into button to step inside the block that calls the subroutine. The subroutine window automatically opens, and you can continue to step through blocks or lines inside the subroutine. As you step through blocks, subroutines, and charts, the tabs at the bottom of the window let you know where you are (see "Using Tabs to View Open Windows" on page 3-15).

You can also set breakpoints on any subroutine block as needed. See "Setting and Removing Breakpoints" on page 7-18 for more information.

Viewing Subroutines

Since subroutines appear on the Strategy Tree, it is easy to view information about them. A subroutine appears in two places on the Strategy Tree: in the Subroutines Included folder and in the folder for the chart that calls it.



In the Subroutines Included folder

In the calling chart folder

You can view, add, and change variables in a subroutine from the Strategy Tree, just as you would for a chart.

Viewing All Subroutines in a Strategy

To see all the subroutines in a Strategy, double-click the Subroutines Included folder on the Strategy Tree. All subroutines associated with the strategy are listed in the Subroutine Files dialog box. Click and drag the right side of the box to see all the information. The path, filename, and reference count (how many times the strategy refers to the subroutine) are shown for each subroutine.

Printing Subroutines

For steps to print a subroutine's graphics, see "Printing Chart or Subroutine Graphics" on page 7-23. To view and print instructions in the subroutine's blocks, see "Viewing and Printing Strategy or Subroutine Elements" on page 7-26.

APPENDIX A

ioControl Troubleshooting

This appendix provides general tips for resolving problems you may encounter while running ioControl or communicating with your hardware. If you are running Windows NT or Windows 2000 and encounter problems with permissions, see page A-9.

For information about types of errors and lists of error codes, see Appendix B, “ioControl Errors and Messages.”

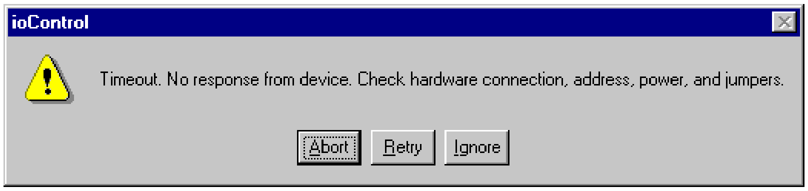
Also check troubleshooting information in your controller’s user guide and the troubleshooting appendix in Opto 22 form 1460, the *SNAP Ethernet-Based I/O Units User’s Guide*.

How to Begin Troubleshooting

You’ve built your strategy, but now you get errors when you try to download it, or it won’t run properly. How do you begin to figure out what’s wrong? The problem may be in communication with the control engine, in communication between the control engine and I/O, in a command, or in the strategy logic. Following are some steps to help you discover the cause.

1. Read Any Error Message Box

If an error message box appears on the computer running ioControl, it’s probably an ioControl error. Here’s an example of an ioControl error message:



Timeout. No response from device. Check hardware connection, address, power, and jumpers.

2. Check Communication with the Control Engine

If there is no error message box, or the error indicates that there may be a communication problem, check whether the PC running ioControl is communicating with the control engine. See "Checking Communication with the Control Engine" on page A-4.

3. Check the Message Queue

If communication with the control engine is OK, check the message queue. To open the queue, see "Inspecting Control Engines and the Queue" on page 5-12. In the "List of Common Messages" on page B-3, look up any errors you find in the queue. Errors are listed in numerical order. Queue errors may indicate problems with a command or with communication to I/O. Check the possible causes for help in fixing problems.

- For help with a command, look up details about the command in the *ioControl Command Reference* or online help.
- For help with communication to I/O, see "Resolving Communication Problems" on page A-5. Many of these suggestions apply to I/O as well as to control engines.

4. Check Status Codes in Your Strategy

If all is well up to this point, double-check Status Codes in your strategy. Status Codes are responses to a command that appear in a variable within your ioControl strategy or as returned values in OptoScript. Your strategy should routinely check status codes and adjust logic as necessary to respond.

Status codes may indicate problems with a command or communication to I/O, or they may indicate a problem in the strategy logic. See "List of Common Messages" on page B-3 for more information. Again, look at the possible causes for help in fixing problems.

5. Call Product Support

If you cannot find the help you need in this book, the *ioControl Command Reference*, or the *SNAP Ethernet-Based I/O Units User's Guide*, call Opto 22 Product Support. See "Product Support" on page 1-4 for contact information.

Strategy Problems

If You Cannot Delete an Item

Sometimes when you try to delete an item in a strategy—a variable, a chart, an I/O unit or point—you receive a message saying "You cannot delete an item with a reference count greater than zero." This message means you cannot delete the item because other elements in the strategy use it.

You can use Find to locate all references to the item you want to delete. For help in using Find, see "Searching" on page 7-30.

Sometimes the reference counts can become incorrect due to cutting and pasting variables or importing charts into a strategy. If a reference count appears to be incorrect, you can rebuild the strategy database by following these steps:

1. Click in the Strategy Tree to make it the active window.
2. Press CTRL+R.
3. Choose Compile➞Compile All.
4. Choose File➞Save All.

   The strategy database is rebuilt and the reference counts should be correct.

If You Have Memory Problems

Control engine memory is allocated as shown in the following table. You can see the total amount of RAM and the amount of battery-backed RAM available for use by inspecting the control engine in Debug mode (see page 5-12).

	SNAP-PAC-S1	SNAP PAC R-series	SNAP-LCE	SNAP Ultimate I/O
Total memory (RAM)	32 MB	16 MB	16 MB	16 MB
Memory used for control	32 MB	10 MB	16 MB	8 MB
Memory available for strategy and variables	16 MB	5 MB	8 MB	5 MB
Total battery-backed RAM	8 MB	2 MB	512 KB	512 KB
Battery-backed RAM for control*	8 MB	1 MB	512 KB	288 KB

* Stores persistent variables, variables initialized on download, autorun flag, and strategy archive. Note that strategies are not automatically saved in battery-backed RAM. Save your strategy to flash memory so it will be available if power is lost to the control engine. See "Saving a Strategy to Flash" on page 7-4.

In general, if you experience memory problems, you can reduce the amount of memory needed by checking strings and tables for lengths and widths that are longer than necessary.

If you are using subroutines, use the minimum number of variables and size of tables that the process requires. Also, less memory is used if only one chart in the strategy calls subroutines than if multiple charts call subroutines.

Since the battery-backed RAM contains variables initialized on download, if you have a large number of these on a smaller controller, you can run out of persistent RAM. To avoid this problem, use as few persistent variables as possible and initialize all other variables on strategy run.

Although you can archive the strategy, the currently running strategy is not stored in battery-backed RAM. To make sure the strategy will run after a power loss, save the strategy to flash memory after downloading it. See page 7-4 for information on saving to flash.

Archiving Strategies

Strategies are archived to battery-backed RAM, which is limited to 256KB. In addition to an archived strategy, battery-backed RAM holds persistent variables and variables that are initiated on download. If you have an unusually large strategy or large numbers of persistent variables or variables that are initialized on download, you may not have sufficient space for an archived strategy. See page 7-5 for more information on archiving.

Do You Use Online Mode?

If you frequently use Online mode to change your strategy, you may find you are having memory problems. When you change a chart in Online mode, a new copy of that chart is downloaded to the control engine, but the old one is not deleted. After you have made a few online changes, these additional chart copies begin to take up memory.

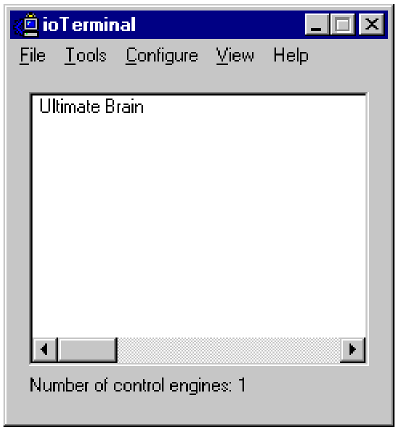
To avoid memory problems, stop the strategy after making several online changes. Completely compile and download the strategy, and old chart copies will be cleared from memory.

Checking Communication with the Control Engine

You can test communication with the control engine by using the ioTerminal utility.

1. From the Start menu, choose Programs→Opto 22→ioProject Software→Tools→ioTerminal.

   The ioTerminal window appears, showing all control engines configured on your system:



Ultimate Brain

Number of control engines: 1

2. If no control engine is listed, configure one by choosing Configure→Control Engine and following directions on the screen. See "Configuring Control Engines" on page 5-1 for help.

3. To verify that a control engine in the list is communicating, double-click the control engine's name.

   The Comm. Loop Time (communication time) in the Inspect Control Engine dialog box indicates how long it takes to gather the information in the dialog box and is a good relative indicator of communication time.

   This dialog box also shows the status of the current strategy and any errors in communication with the control engine. For further explanation, see "Inspecting Control Engines and the Queue" on page 5-12.

4. If you receive an error indicating a communication problem, go on to the next section.

Resolving Communication Problems

Matching ioControl Configuration to the Real World

I/O unit and point configuration in ioControl must match actual I/O units and points with which the control engine is communicating. See brain and I/O module data sheets for specifications and information, and see page 6-4 for help configuring I/O in ioControl.

Resolving TCP/IP Cannot Connect Errors (-412)

Many problems with Ethernet connections return a TCP/IP Cannot Connect error. Cannot connect errors are probably the most common communication problem with control engines. They indicate that a TCP/IP connection could not be made to the control engine within the specified time interval.

If you receive this error, first check the following:

- Make sure the control engine has been turned on.
- Verify that the correct IP address appears for the control engine.
- Make sure your control engine has been assigned a valid IP address. These controllers and brains come from the factory with a default IP address of 0.0.0.0, which is invalid. For help in assigning an IP address, see the *ioManager User's Guide*.
- Make sure you have up-to-date drivers installed on your computer's Network Interface Card (NIC). Contact your system administrator or the manufacturer of the card for help.
- If problems persist, you can increase the length of time before a timeout occurs. Choose Configure➞Control Engines and change the Timeout (mSec) field to a larger number.

Pinging the Control Engine

If you still cannot communicate with the control engine after you have checked these items, try to reach it using the PING protocol.

Choose Start➞Programs➞MS-DOS Prompt. At the prompt, type:

```
ping [control engine’s IP address]
```

(For example, type `ping 10.192.54.40`.)

If data is returned from the control engine, it can be found on the network.

If the PING command cannot be found—Verify that the PC has TCP/IP bound to and configured on the network adapter.

If you are running Windows 95 or Windows 98, follow these steps:

a. Choose Start→Settings→Control Panel and double-click Network.

b. Highlight the adapter in the list. Make sure both NetBEUI and TCP/IP appear just below the name of the adapter. Click Properties.

c. Highlight TCP/IP and click Properties. Verify that the IP address and subnet mask are appropriate for your network.

If you are running Windows NT, follow these steps:

a. Choose Start→Settings→Control Panel and double-click Network.

b. Click the Protocols tab. Make sure both NetBEUI and TCP/IP are listed. Highlight TCP/IP and click Properties.

c. Highlight the adapter name in the list. Verify that the IP address and subnet mask are appropriate for your network.

If you are running Windows 2000, follow these steps:

a. Choose Start→Settings→Control Panel and double-click Network and Dialup Connections.

b. Right-click your network card and choose Properties from the pop-up menu. Make sure that TCP/IP is present and checked.

c. Highlight TCP/IP and click Properties. Verify that the IP address and subnet mask are appropriate for your network.

If you see the message “Destination host route not defined,” the control engine probably has an IP address and subnet mask that are incompatible with those on the computer. Subnetwork numbers and netmasks must be identical on the control engine and the computer.

If you see the message “No response from host,” check the following:

- Are the computer and control engine correctly connected? Is the control engine turned on?
- Are the IP address and subnet mask on the control engine compatible with those on the computer?

If your host computer has more than one Ethernet card, check your route table to make sure packets are routed to the correct adapter card.

If all else fails, connect the PC and the control engine using an Ethernet crossover cable, and retest the connection.

If you still cannot ping the control engine, contact Product Support. (See page 1-4.)

Other Troubleshooting Tools

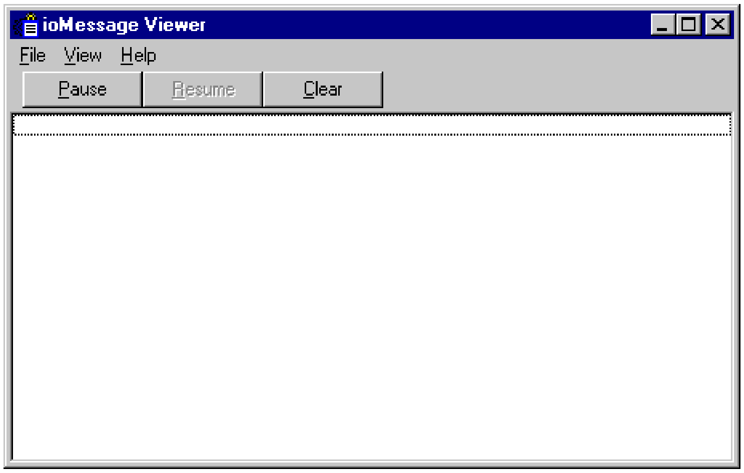
Checking Detailed Communication Information Using ioMessage Viewer

For detailed information about each communication transaction, use the ioMessage Viewer utility.

1. In the Start menu, choose Programs→Opto 22→ioProject Software→Tools→ioMessageViewer.

   You can also start ioMessage Viewer from ioTerminal by choosing Tools→Start ioMessage Viewer, or from ioControl in Debug mode by choosing Debug→Sniff Communication.

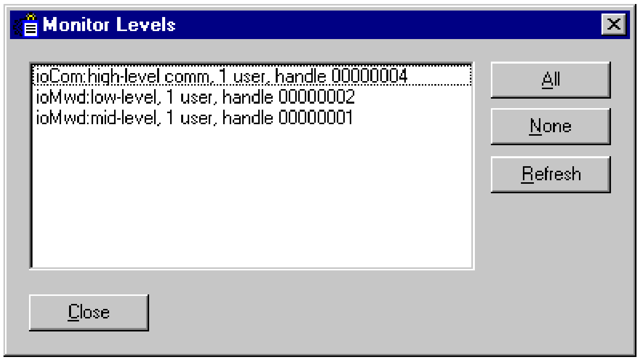
   The ioMessage Viewer window appears.



   In most cases the main window is blank, indicating that no messages are being monitored between the PC and the active control engine. In some cases, for example when ioControl launches ioMessage Viewer, messages should appear immediately.

2. To start monitoring or change the level of monitoring, choose View→Monitor Levels.

The Monitor Levels dialog box lists all the possible levels to monitor. You can click Refresh to make sure the list is up to date.



ioCom:high-level comm, 1 user, handle 00000004
ioMwd:low-level, 1 user, handle 00000002
ioMwd:mid-level, 1 user, handle 00000001

3. Highlight one or more of the monitor levels in the list, and click Close.

   You return to the ioMessage Viewer window, where the changes you made are reflected at once. To stop monitoring, click Pause. To start monitoring again, click Resume. To erase all messages from the window, click Clear.

   By default, communication messages in ioMessage Viewer are automatically saved to a log file named IOSNIF.LOG. You can toggle saving on and off by choosing File→Log to File.

   Also by default, messages are temporarily stored in system cache memory before being saved to the log file. If you are having trouble with system crashes and need to capture messages just before a crash, however, you can choose File→Flush File Always to send messages directly to the log file.

4. To view or edit the log file, choose File→Edit Log File.

   The file opens in a text editor. Logging is turned off when you open the file.

5. View or edit the file as needed, and then close it.

   You return to the ioMessage Viewer window.

6. To resume logging, choose File→Log to File.

7. To rename the log file or change its location, choose File→Select Log File. Navigate to the location where you want to save the file and enter a name. Click OK.

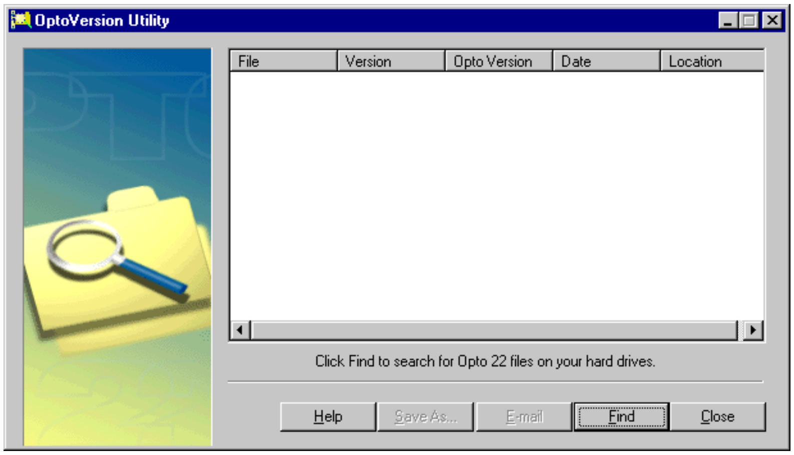
8. When you have finished monitoring communication, close the ioMessage Viewer window.

   If you leave it open, it will normally appear on top of other running programs. If you don’t want it to appear on top of other programs, choose View→Always on Top to toggle that option.

Checking File Versions for Opto 22 Software

Sometimes problems may be caused by older or misplaced files. Product Support may ask you to run OptoVersion to check the versions and paths of your Opto 22 .dll and .exe files. Here's how:

1. From the Start menu, choose Programs➞Opto 22➞ioProject Software➞Tools➞OptoVersion.



Click Find to search for Opto 22 files on your hard drives.

2. In the OptoVersion window, click Find.

   The utility searches your hard drive and prints a list of Opto-related files found.

3. To see more information on any file, double-click its name. To sort the list in a different order, click any column heading.

4. To email the information to Opto 22 Product Support, click E-mail.

   The utility saves the list to a file named Version.bd in the same directory that contains OptoVersion.exe. If you use Microsoft Outlook as your email program, a new message automatically appears addressed to Product Support, with the version file attached.

5. If you use Microsoft Outlook, add comments to the new message and click Send.

6. If you use another email program, attach the Version.bd file to an email message and address the message to **support@opto22.com**, along with an explanation of the problem you're experiencing.

7. To save the file, click Save As. Give the file a name and save it in the location you want.

   OptoVersion also creates a tab-delimited file with the same file extension and in the same directory. This file has the same name you gave it but with `_Delimited` added. For example, if you name the saved file `Opto_software.txt`, the tab-delimited file is named `Opto_software_Delimited.txt`. This file can be opened in Microsoft Excel or other programs to easily sort and view its contents.

Problems with Permissions in Windows 2000

When you set up controllers on a computer running the Microsoft Windows 2000 operating system, typically you are using the computer with top-level "administrator" privileges. If someone later uses this same computer to run ioControl or ioDisplay, but logs in to the computer with lower-level, non-administrator privileges, the application may not recognize control engines that have been previously configured.

If this problem occurs, you can modify the permissions to let specific users access previously configured control engines without having administrator access. This is done using the Registry Editor utility. Follow the steps below.

WARNING: *Use the Windows Registry Editor carefully. It is strongly recommended that you make a backup copy of your Windows Registry before continuing with this procedure. Without a backup copy, if you delete the wrong properties and cannot return the Registry to its original state, application and system files can become unusable and will have to be reinstalled.*

1. From the Windows Start menu, select Run.

   The Run dialog box appears.

2. Enter the following command in the Open field and press ENTER:

   ```
   regedt32
   ```

   NOTE: This is NOT regedit.exe, which is a similar tool.

   The Registry Editor main window appears with several open windows inside it.

3. Select the HKEY_LOCAL_MACHINE window to make it active.

4. Double-click the Software folder in the HKEY_LOCAL_MACHINE window.

5. Select the Opto22 folder.

6. In the Security menu, choose Permissions.

   The Registry Key Permissions dialog box opens. Make sure that "Opto22" appears next to Registry Key at the top of the window.

7. Click Add.

8. In the Select Users, Computers, or Groups dialog box, select the name of the appropriate group or domain from the Look In drop-down list.

9. In the Name list, select the name of the user or group that should have control engine access and then click Add.

10. If it is not already selected, check "Full Control" in the Permission area. Make sure "Allow inheritable permissions from parent to propagate to this object" is checked.

11. Click OK.

12. Select Registry→Exit to close the Registry Editor.

13. Restart the computer.

The user or group you added can now use control engines without having administrator access.

APPENDIX B

ioControl Errors and Messages

Introduction

This appendix discusses errors and messages you may see in ioControl and their possible causes. Errors and messages may appear with text only or with a negative number and text. The more common errors and messages are listed in this chapter in numeric order, starting on page B-3.

See "ioControl Troubleshooting" on page A-1 for additional help in resolving errors.

Types of Errors

As you work in ioControl, you may see three types of errors:

- ioControl Errors appear in dialog boxes on the computer screen.
- Queue Messages (both errors and other messages) appear in the control engine's message queue.
- Status Codes appear in variables or as returned values in OptoScript.

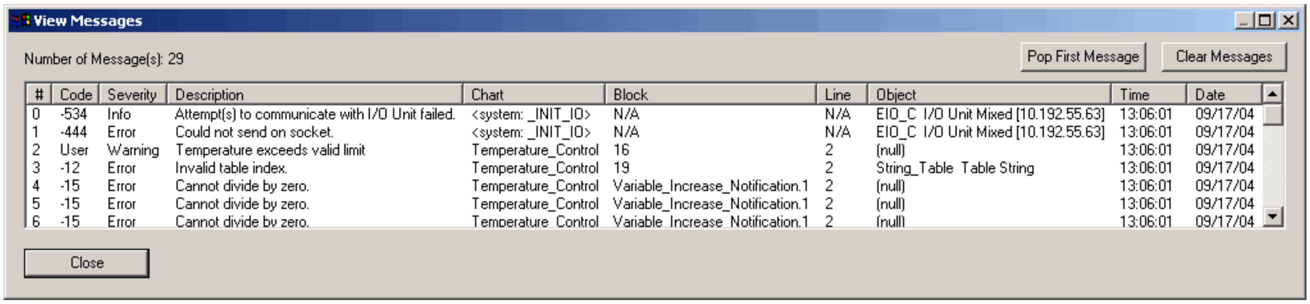
ioControl Errors

ioControl errors indicate a problem within ioControl that may have been reported by the control engine or may have occurred before control engine communication.

ioControl errors appear in dialog boxes on the computer running ioControl. Some of these errors appear as numbers, some as text, and some show both numbers and text. An example of an ioControl error is "Timeout. No response from device." Another example is "TCP/IP: Cannot connect error" with an error code of -412.

Queue Messages

Queue messages indicate an error or other message during Strategy operation, and they appear in the ioControl message queue. (For information on viewing the queue, see “Inspecting Control Engines and the Queue” on page 5-12.) Here’s an example of a message queue:



Number of Message(s): 29

#	Code	Severity	Description	Chart	Block	Line	Object	Time	Date
0	-534	Info	Attempt(s) to communicate with I/O Unit failed.	<system: _INIT_IO>	N/A	N/A	EIO_C I/O Unit Mixed [10.192.55.63]	13:06:01	09/17/04
1	-444	Error	Could not send on socket.	<system: _INIT_IO>	N/A	N/A	EIO_C I/O Unit Mixed [10.192.55.63]	13:06:01	09/17/04
2	User	Warning	Temperature exceeds valid limit	Temperature_Control	16	2	(null)	13:06:01	09/17/04
3	-12	Error	Invalid table index.	Temperature_Control	19	2	String_Table Table String	13:06:01	09/17/04
4	-15	Error	Cannot divide by zero.	Temperature_Control	Variable_Increase_Notification.1	2	(null)	13:06:01	09/17/04
5	-15	Error	Cannot divide by zero.	Temperature_Control	Variable_Increase_Notification.1	2	(null)	13:06:01	09/17/04
6	-15	Error	Cannot divide by zero.	Temperature_Control	Variable_Increase_Notification.1	2	(null)	13:06:01	09/17/04

This queue shows several types of messages that you might see. To see all the information in a column, drag the edge of the column heading to the right.

Code Queue errors generated by the system are shown as negative numbers in the Code column. For example, if you specify a table index that is greater than the number of elements in the table, an error -12, “Invalid table index,” appears, as in message #3 above. Common queue errors for each command are listed in the *ioControl Command Reference* and in online help.

If the Code column indicates *User*, the error is one you have placed in the queue using the command Add Message to Queue. User messages can help with troubleshooting. Message #2 above is an example of a message the user placed in the Temperature_Control chart.

Severity The Severity column indicates the type of message: information, warning, or error.

Chart, Block, Line, Object If an ioControl command in the strategy caused an error, the chart name, block number, and line number (if you are in Full Debug mode) where the command appears are listed. Message #3 above is an example: an invalid table index was used in block 19 of the Temperature_Control chart. The Object column shows the table name.

If an error did not occur in a strategy chart, the Chart column shows <system>. Messages 1 and 2 occurred when the strategy was unable to initialize an I/O unit, so the Chart column shows <system: _INIT_IO>. Block and Line do not apply, but the Object column shows the name and IP address of the I/O unit.

If an error was caused by a subroutine, the Chart column shows the name of the chart that calls the subroutine, and the Block column shows the name of the subroutine and the block number where the error occurred, in the format `<subroutine name>.<block number>`. Messages 4, 5, and 6 are examples; these errors occurred in block 1 of the subroutine Variable_Increase_Notification, which was called by the Temperature_Control chart.

Using Queue Messages

If a block number is listed for the error, look in that block in the Strategy to find the ioControl command that caused the error. The easiest way to find a block is to open the chart or subroutine, then choose Center on Block from the View menu. You can click the Block column to sort the blocks by number and locate the one with the problem.

To see which line within a block is causing the error, in ioControl Configure mode, choose Configure→Full Debug. When the error appears in the queue, it will include the line number of the command as well as the block ID.

Status Codes

Status Codes indicate the success or failure of an ioControl command (instruction), and they are reported by the control engine to a variable in your ioControl Strategy or as a returned value in OptoScript. The status code is either zero (indicating a successful command) or a negative number (indicating an error).

For example, suppose you use the command Transmit Numeric Table. You create a variable named Transmit_Status to put the status of the command in. In Transmit_Status you receive either a zero, indicating that the table was successfully transmitted, or you receive a negative number such as -37 or -42.

Status codes that may be returned are listed for each command in the *ioControl Command Reference* and in online help.

List of Common Messages

The following messages may appear in ioControl. They are listed in numeric order.

If an X appears in the Q? column, the code number appears in the message queue. If an X appears in the I/O? column, the message is an I/O unit error and may appear either in the message queue, as a status code in a variable, or both. For more information on handling I/O unit errors, see "Error Handling Commands" on page 10-56 and "Handling I/O Errors Efficiently" on page 4-23.

#	Description	Possible Cause	Q?	I/O?
0	Operation performed successfully.	NOT AN ERROR. Indicates the command was successful.		
-1	Undefined command.	An unknown command was sent to the ioControl engine.		
-2	Checksum or CRC mismatch.	When comparing DVFs (Data Verification Fields), a mismatch occurred. Examples of DVFs include checksum and CRC.		
-3	Buffer overrun or invalid length error.	In string manipulations, a string was requested that is longer than the string it will be put into, or the destination string length <= 0.	X	X

#	Description	Possible Cause	Q?	I/O?
-4	Device has powered up. ('Powerup clear expected' message received.)	NOT AN ERROR. The device has been turned off and then on again since the last communication, and is now ready.	X	
-5	Operation failed.	An attempt to store the strategy to flash failed, or an attempt to do something with a chart failed (like call, continue, suspend, start, initialize threads).		
-6	Data field error.	Invalid year entered (must be between 2000 and 2099), or invalid data read from memory when attempting to read the strategy from flash memory.	X	
-7	Watchdog timeout has occurred.	See "Add I/O Unit Dialog Box" on page 6-13.		
-8	Invalid data.	Invalid data read when attempting to read a strategy from flash, or an invalid character number was passed to a string function.		
-10	Invalid port number.	Valid range for Ethernet is 0–65535. For a communication handle, serial port format may be incorrect.		X
-11	Could not send data.	A Transmit or Transfer command fails when using a comm handle. For example, an attempt is made to send a file to a remote ftp server that has gone off-line.		
-12	Invalid table index.	Used an index greater than the number of elements in the table.	X	X
-13	Overflow error.	Typically, a math result too big to fit in the value passed, while doing ***number conversion functions*** such as converting a float value to engineering units, a float to an unsigned integer, a float to an integer, a 64-bit integer to a 32-bit integer, a floating point value to a 64-bit integer, a floating point value to an unsigned 64-bit integer, an ASCII value to a float. Also ***math functions*** such as multiply, exponentiation, hyperbolic sine, hyperbolic cosine, function x to the y, add, subtract, modulo, negate, move/assign. ***Time/date functions***: setting a month not in the range 1–12, a day not in the range 1–31, hours not in 0–23, minutes 0–59, seconds 0–59. Number entered on host port (for example, through ioTerm) was too big for data types.	X	
-14	Invalid number.	Math resulted in an invalid number (like infinity or an imaginary number): natural log of the floating point number, square root, arc sine, arc cosine of float, function x to the y (with negative x).	X	
-15	Cannot divide by zero.	Attempted to divide a number by zero.	X	
-16	Bus error.	Contact Product Support. See page 1-4.		
-17	Port already locked on ioControl engine.	Attempted to lock a connection that's already locked.		X
-20	Device busy. May be in use by another user or another application.	A resource is already acquired by another task or process.	X	
-21	Had to relock host port in 'QUIT'.	Host port needed relocking.		

#	Description	Possible Cause	Q?	I/O?
-23	Destination string too short.	In string manipulations, a string was requested that is longer than the string it will be put into, or destination string length <= 0.	X	X
-25	Port not locked.	Attempted to transmit or receive on a connection that wasn't locked, or to unlock a connection that wasn't locked.		X
-26	Unknown response from device.	OptoMMP-based protocol packet returned by the device was invalid.		X
-29	Wrong object type. Most likely caused by moving a pointer table element to a pointer of the wrong type.	An object was passed to a command that doesn't handle that object type.		
-34	Invalid I/O command or invalid memory location.	Contact Product Support. See page 1-4.		X
-35	Point mismatch	An analog point is incorrectly configured (for example, an output point is configured as an input point).	X	X
-36	Invalid command or feature not implemented.	Feature not yet implemented for this hardware/command combination, or command may not apply to the type of communication handle you are using.		X
-37	Timeout on lock.	Unable to lock a resource (such as a variable) for exclusive writing within the timeout period.		X
-38	Timeout on send.	Unable to send communication in the timeout period.		
-39	Timeout on receive.	Unable to receive communication in the timeout period.		
-42	Invalid limit (on string index, task state, priority, etc.).	A character number greater than the length of the string was used (or the string had a zero length), or an invalid value was passed when setting the state of a chart (running, etc.)		
-44	String too short.	String less than 8 characters used to read time ("hh:mm:ss"), or not 8 or 10 characters for a date, or zero length on a string function.	X	
-45	Null string.	Attempted to use an uninitialized string.		
-46	Invalid string.	String not 8 characters long when setting time, or invalid format when setting date, or invalid communication handle string or comm handle command (such as a missing colon).	X	
-47	Invalid connection. Device drivers might be missing or not loaded/running.	Attempted to open an already open connection.		X
-49	No more connections are available. Maximum number of connections already in use.	No more sessions available for Ethernet (maximum is 32).		X
-50	Open connection timeout. Could not establish connection within the timeout period.	Unable to open a connection in time.		X
-52	Invalid connection—not opened.	Attempted to close a connection that wasn't opened. Communication handle may have been closed by a previous command that failed.		X
-57	String not found.	Substring not found in the string being searched.		

#	Description	Possible Cause	Q?	I/O?
-58	No data received. or Character not found.	Attempted to read an empty buffer (or a connection with no characters waiting); or the I/O unit may be turned off or unreachable; or when searching a string for a particular character, the character wasn't found.		X
-59	Could not receive data.	Command may not apply to the type of communication handle you are using; for example, Receive commands cannot be used with ftp comm handles. (Use the *get* option with Send Communication Handle Command instead.)		
-60	Empty stack error. ioControl engine attempted to perform an operation that expected data on the ioControl engine stack.	Contact Product Support. See page 1-4.		
-61	Dictionary full error. ioControl engine dictionary is full and no more 'words' can be defined. Clear ioDisplay words using ioDisplay or ioTerminal, if appropriate.	Attempted to create a variable, command, or similar item when there's no room left. Or large string or numeric tables are being created in the strategy.		
-62	Stack full error. ioControl engine stack has grown too big.	The ioControl engine stack is full.		
-64	Execute-only error. A command or 'word' was encountered that cannot be used when compiling.	Contact Product Support. See page 1-4.		
-66	Requested item in protected dictionary	Attempted to remove a strategy when there was no strategy in the control engine.		
-67	Out of memory. To minimize the size of your strategy, reduce the number and size of variables (especially tables). You can also shrink your strategy by using subroutines to perform common tasks	No room left to create any variables or save any data on the stack. Or an attempt was made to save the strategy to flash, without enough room in flash to save it.	X	
-69	Invalid parameter (null pointer) passed to command.	Attempted to use an uninitialized pointer, or a null pointer was received by a command.	X	
-70	Not enough data supplied.	Table index given is larger than the size of the table.		X
-71	Out of persistent memory. If applicable, check length of tables.	Too many persistent variables, variables initialized on download, or too large a strategy archive to fit in battery-backed RAM.		
-93	I/O unit not enabled. Previous communication failure may have disabled the unit automatically. Reenable it and try again.	I/O unit may have been disabled by a communication failure that happened earlier.		X
-103	Port could not be unlocked. Task attempting to unlock the port does not match the task with the current lock on that port.	Attempted to unlock a connection that was locked by a different task.		X
-203	Driver could not be found or loaded.	Communication command didn't find the driver described in the communication handle. Make sure driver name is not misspelled (for example, tcp must be lower case).		X
-407	File not found.	Attempted to save a strategy to flash, but there was no strategy in RAM to save.		
-412	TCP/IP: Cannot connect error	Ethernet "connect" failed. See page A-5.		X

#	Description	Possible Cause	Q?	I/O?
-417	Cannot open file	File does not exist or filename may be incorrect.		
-430	Invalid data range. Verify high value is greater than low value.	Invalid data passed to driver (for example, a point number larger than the maximum number of points on the rack).		X
-433	Object/device already locked.	Couldn’t set the state of a chart (running, suspended, etc.) because it’s already locked by something else.		
-437	No acceptable socket interface found.	Ethernet “accept” attempted, but no more sessions are available (32 total).	X	
-438	Could not create socket.	Attempt to create a new Ethernet socket failed.		X
-442	Could not accept on socket.	Ethernet “accept” failed.		X
-443	Could not receive on socket.	Ethernet “receive” failed.		X
-444	Could not send on socket.	Ethernet “send” failed.		X
-531	Buffer full.	Attempted to write to a full buffer. For a serial communication handle, data is being sent faster than the serial port can send and buffer it. Use a faster baud rate or a delay between Transfer/Transmit commands.		X
-534	Attempts to communicate with I/O unit failed.	I/O unit may have lost power or network connection.	X	X
-700	PID Loop has been configured outside of this strategy and could conflict with this strategy’s logic.	You are trying to download a new strategy, but a PID loop is currently running on the brain. Open ioManager and turn off the loop in Inspect mode by changing its algorithm to None.		
-8607	Invalid protocol.	Attempted to set a port to an unknown mode.		X
-8608	Port initialization failed.	While starting up a chart or task, the default host port could not be created.	X	
All -10,000 and -11,000 errors	[Various descriptions]	Socket or network problems. Check cables and connections to control engine; cycle power to control engine.		

APPENDIX C

ioControl Files

Introduction

This appendix lists all of the ioControl file types and special files. You can use this information as a reference to determine what types of files are present in your ioControl project directory when you're looking through your ioControl or project directory.

Files Related to a Strategy

<strategy>.idb	ioControl strategy database
<strategy>.crn	Run file (compiled file that is sent to the control engine)
<strategy>.crn1	Intermediate run file (component of the run file)
<strategy>.crn2	Intermediate run file (component of the run file)
<strategy>.crn3	Intermediate run file (component of the run file)
<strategy>.inc	Initialization data for variables with "Init on Download" option (component of the run file)
<strategy>.inf	Strategy configuration information
<strategy>.$idb	Temporary ioControl strategy database file
<strategy>.lidb	ioControl strategy database lock file
<strategy>.per	Persistent variable definitions
<strategy>.<control engine>.cdf	Control engine download file for special circumstances (see page 7-9)
<chart name>.cht	Chart
<chart name>.ccd	Compiled chart code (component of the run file)
<chart name>.con	Online compiled chart code

<chart name>.cxf	Exported chart file
<filename>.wth	Watch window file (you name the file)
<filename>.otg	Exported I/O configuration file (you name the file)
<strategy.date.time>.zip	Strategy archive file (automatically named; see "Archiving Strategies" on page 7-5 for file name formats)

Files Associated with a Subroutine

<subroutine name>.isb	Subroutine
<subroutine name>.ini	Subroutine configuration information
<subroutine name>.isc	Compiled subroutine (component of the run file)
<subroutine name>.lisb	Subroutine lock file

Files in the ioControl Directory

<xxx>.io.def	Object definition files (commands, I/O points, and I/O units). You must not modify these files.
ioCtrl.exe	ioControl executable file
ioControl.cnt	ioControl help contents file
ioControl.GID	ioControl help support file (created when you launch the help file)
ioControl.hlp	ioControl help file
ioControlCommands.cnt	Commands help contents file
ioControlCommands.GID	Commands help support file (created when you launch the help file)
ioControlCommands.hlp	Commands help file
ioCtrlTools.dat	File that lists software applications you've configured in the Tools menu to launch from ioControl
ioSnif.log	ioMessage Viewer log file
OptoScriptTemp.txt	A temporary file
Readme.txt	README text file containing information about ioControl

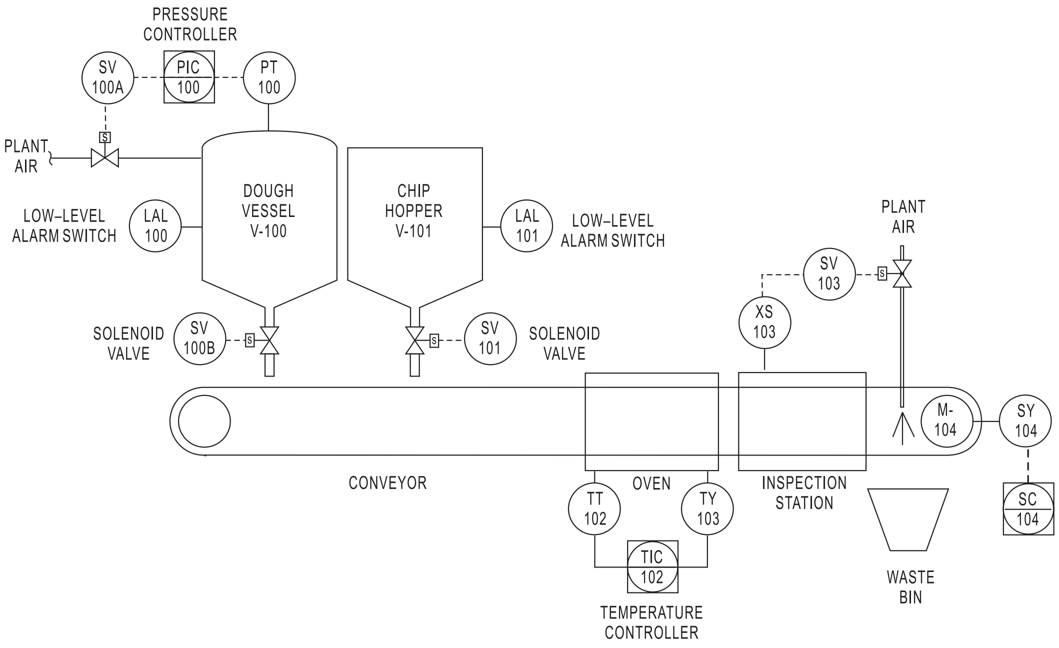
Sample Strategy

Introduction

Chapter 2, “ioControl Tutorial” introduced the Cookies strategy, a sample project used to illustrate how ioControl works. Although this strategy is based on a mythical factory, you may want to know more about the factory, its process, and its hardware. This appendix gives you that information.

Factory Schematic

The following schematic drawing summarizes the cookie factory:



Description of the Process

Dough Vessel

The first station in our process is the dough vessel. This tank contains a pre-made cookie dough mix.

Dough is dispensed onto the conveyor belt through a valve (SV-100B) at the bottom of the vessel. The dough, being somewhat viscous, must be kept under low pressure to dispense properly. To monitor the pressure, we have included a pressure transmitter (PT-100) in the vessel. Our control engine (a SNAP Ultimate I/O system) maintains the vessel pressure through a plant air valve (SV-100A).

The vessel also includes a level switch (LAL-100) to tell us when the dough level is low. When it is, the process is halted so that an operator can refill the vessel.

Chip Hopper

The chip hopper supplies chocolate chips. A chip dispenser valve (SV-101) controls the number of chips dropped on each cookie. Like the dough vessel, this tank also includes a level switch (LAL-101) to stop the system when the chip hopper needs refilling.

Oven

After the dough and chips have been dropped onto the conveyor, the conveyor sends the cookie into the oven, and the oven bakes it.

Inspection Station

Our freshly baked cookies then move to the inspection station, where someone inspects them. If the cookie does not fall within normal tolerances— for example, it doesn't have enough chips or is shaped oddly—the inspector closes a switch (XS-103), signalling the bad cookie. A valve (SV-103) then opens to allow plant air to blow the reject cookie into a waste bin.

If the cookie passes the inspection, it moves on to packaging and shipping.

Conveyor

The conveyor and its motor continuously move the cookies from the dough vessel to the inspection station. The conveyor speed is controlled through an analog output (SY-104) from a speed controller (SC-104).

Emergency Stops

Wired at key locations around our bakery are emergency-stop buttons. If something goes wrong with the process, an operator can press any of these E-STOP buttons.

The buttons are wired in series and are normally closed, so pressing any E-STOP button breaks the circuit. One digital input can monitor all the buttons. The system can be restarted by resetting the button.

Required I/O

Here's the list of analog and digital I/O modules required for the cookie factory:

Analog I/O

Name	Description	Type	Module	Range
PT-100	Dough Vessel Pressure	Input	SNAP-AIV (–10 to +10 VDC)	0–15 psig
TT-102	Oven Temperature	Input	SNAP-AICTD (ICTD)	-50–350°C
TY-102	Oven Temperature Control	Output	SNAP-AOV-27 (–10 to +10 VDC)	0–100%
SY-104	Conveyor Speed Control	Output	SNAP-AOV-27 (–10 to +10 VDC)	0–100%

Digital I/O

Name	Description	Type	Module	States
SV-100A	Pressure Control Valve	Output	SNAP-ODC5SRC (5–60 VDC)	0=Closed 1=Open
SV-100B	Dough Dispense Valve	Output	SNAP-ODC5SRC (5–60 VDC)	0=Closed 1=Open
LAL-100	Dough Level Alarm	Input	SNAP-IDC5D (2.5–28 VDC)	0=OK 1=Low
SV-101	Chip Dispense Valve	Output	SNAP-ODC5SRC (5–60 VDC)	0=Closed 1=Open
LAL-101	Chip Level Alarm	Input	SNAP-IDC5D (2.5–28 VDC)	0=OK 1=Low
XS-103	Inspection Signal	Input	SNAP-IDC5D (2.5–28 VDC)	0=OK 1=Reject
SV-103	Reject Valve	Output	SNAP-ODC5SRC (5–60 VDC)	0=Closed 1=Open
XS-105	Emergency Stop	Input	SNAP-IDC5D (2.5–28 VDC)	0=Stop 1=OK

APPENDIX E

OptoScript Command Equivalents

Introduction

This appendix compares standard ioControl commands to their equivalents in OptoScript code. See Chapter 11, “Using OptoScript” and Appendix F, “OptoScript Language Reference” for more information on OptoScript.

The following table lists both action and condition commands in alphabetical order. The Type column shows whether the OptoScript command is a function command (F) or a procedure command (P). Function commands return a value from their action; procedure commands do not. For more information on command type, see “More About Syntax with Commands” on page 11-12.

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Absolute Value	`AbsoluteValue(Of)`		F
Accept Incoming Communication	`AcceptIncomingCommunication(Communication Handle)`		F
Add		`x + y`	F
Add Message to Queue	`AddMessageToQueue(Severity, Message)`		P
Add User Error to Queue	`AddUserErrorToQueue(Error Number)`		P
Add User I/O Unit Error to Queue	`AddUserIoUnitErrorToQueue(Error Number, I/O Unit)`		P
AND		`x and y`	F
AND?		See AND	F
Append Character to String		`s1 += 'a';`	P
Append String to String		`s1 += s2;`	P
Arccosine	`Arccosine(Of)`		F
Arcsine	`Arcsine(Of)`		F
Arctangent	`Arctangent(Of)`		F
Bit AND		`x bitand y`	F
Bit AND?		See Bit AND	F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Bit Clear	BitClear(*Item, Bit to Clear*)	x bitand (bitnot (1 << nBitToClear))	F
Bit NOT		bitnot x	F
Bit NOT?		See Bit NOT	F
Bit Off?	IsBitOff(*In, Bit*)	not (x bitand (1 << nBitToTest))	F
Bit On?	IsBitOn(*In, Bit*)	See Bit Test	F
Bit OR		x bitor y	F
Bit OR?		See Bit OR	F
Bit Rotate	BitRotate(*Item, Count*)		F
Bit Set	BitSet(*Item, Bit to Set*)	x bitor (1 << nBitToSet)	F
Bit Shift		x << nBitsToShift	F
Bit Test	BitTest(*Item, Bit to Test*)	x bitand (1 << nBitToTest)	F
Bit XOR		x bitxor y	F
Bit XOR?		See Bit XOR	F
Calculate & Set Analog Gain	CalcSetAnalogGain(*On Point*)		F
Calculate & Set Analog Offset	CalcSetAnalogOffset(*On Point*)		F
Calculate Strategy CRC	CalcStrategyCrc()		F
Calling Chart Running?	IsCallingChartRunning()		F
Calling Chart Stopped?	IsCallingChartStopped()		F
Calling Chart Suspended?	IsCallingChartSuspended()		F
Caused a Chart Error?	HasChartCausedError(*Chart*)		F
Caused an I/O Unit Error?	HasIoUnitCausedError(*I/O Unit*)		F
Chart Running?	IsChartRunning(*Chart*)		F
Chart Stopped?	IsChartStopped(*Chart*)		F
Chart Suspended?	IsChartSuspended(*Chart*)		F
Clamp Float Table Element	ClampFloatTableElement(*High Limit, Low Limit, Element Index, Of Float Table*)		P
Clamp Float Variable	ClampFloatVariable(*High Limit, Low Limit, Float Variable*)		P
Clamp Integer 32 Table Element	ClampInt32TableElement(*High Limit, Low Limit, Element Index, Of Integer 32 Table*)		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Clamp Integer 32 Variable	ClampInt32Variable(*High Limit, Low Limit, Integer 32 Variable*)		P
Clamp Mistic PID Output	ClampMisticPidOutput(*High Clamp, Low Clamp, On PID Loop*)		P
Clamp Mistic PID Setpoint	ClampMisticPidSetpoint(*High Clamp, Low Clamp, On PID Loop*)		P
Clear All Errors	ClearAllErrors()		P
Clear All Event Latches	ClearAllEventLatches(*On I/O Unit*)		P
Clear All Latches	ClearAllLatches(*On I/O Unit*)		P
Clear Communication Receive Buffer	ClearCommunicationReceiveBuffer (*Communication Handle*)		P
Clear Counter	ClearCounter(*On Point*)		P
Clear Event Latch	ClearEventLatch(*On Event/Reaction*)		P
Clear HDD Module Off-Latches	ClearHddModuleOffLatches(*I/O Unit, Module Number, Clear Mask*)		F
Clear HDD Module On-Latches	ClearHddModuleOnLatches(*I/O Unit, Module Number, Clear Mask*)		F
Clear Off-Latch	ClearOffLatch(*On Point*)		P
Clear On-Latch	ClearOnLatch(*On Point*)		P
Clear Pointer		`pn1 = null;`	F
Clear Pointer Table Element		`pt[0] = null;`	P
Close Communication	CloseCommunication(*Communication Handle*)		F
Comment (Block)		`/* block comment */`	P
Comment (Single Line)		`// single line comment`	F
Communication Open?	IsCommunicationOpen(*Communication Handle*)		F
Communication to All I/O Points Enabled?	IsCommToAllIoPointsEnabled()		F
Communication To All I/O Units Enabled?	IsCommToAllIoUnitsEnabled()		F
Compare Strings	CompareStrings(*String 1, String 2*)		F
Complement		`-x`	P
Continue Calling Chart	ContinueCallingChart()		F
Continue Timer	ContinueTimer(*Timer*)		P
Convert Float to String	FloatToString(*Convert, Length, Decimals, Put Result in*)		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Convert Hex String to Number	HexStringToNumber(*Convert*)		F
Convert IEEE Hex String to Number	IEEEHexStringToNumber(*Convert*)		F
Convert Integer 32 to IP Address String	Int32ToIpAddressString(*Convert, Put Result In*)		F
Convert IP Address String to Integer 32	IpAddressStringToInt32(*Convert*)		F
Convert Mistic I/O Hex to Float	MisticIoHexToFloat(*Convert*)		F
Convert Number to Formatted Hex String	NumberToFormattedHexString(*Convert, Length, Put Result in*)		P
Convert Number to Hex String	NumberToHexString(*Convert, Put Result in*)		P
Convert Number to Mistic I/O Hex	NumberToMisticIoHex(*Convert, Put Result in*)		P
Convert Number to String	NumberToString(*Convert, Put Result in*)		P
Convert Number to String Field	NumberToStringField(*Convert, Length, Put Result in*)		P
Convert String to Float	StringToFloat(*Convert*)		F
Convert String to Integer 32	StringToInt32(*Convert*)		F
Convert String to Integer 64	StringToInt64(*Convert*)		F
Convert String to Lower Case	StringToLowerCase(*Convert*)		P
Convert String to Upper Case	StringToUpperCase(*Convert*)		P
Copy Current Error to String	CurrentErrorToString(*Delimiter, String*)		P
Copy Date to String (DD/MM/YYYY)	DateToStringDDMMYYYY(*String*)		P
Copy Date to String (MM/DD/YYYY)	DateToStringMMDDYYYY(*String*)		P
Copy Time to String	TimeToString(*String*)		P
Cosine	Cosine(*Of*)		F
Decrement Variable	DecrementVariable(*Variable*)	`x = x - 1;`	P
Delay (mSec)	DelayMsec(*Milliseconds*)		P
Delay (Sec)	DelaySec(*Seconds*)		P
Disable Communication to All I/O Points	DisableCommuncationToAllIoPoints()		P
Disable Communication to All I/O Units	DisableCommunicationToAllIoUnits()		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Disable Communication to Event/Reaction	DisableCommunicationToEventReaction(*Event/Reaction*)		P
Disable Communication to I/O Unit	DisableCommunicationToIoUnit(*I/O Unit*)		P
Disable Communication to Mistic PID Loop	DisableCommunicationToMisticPidLoop(*PID Loop*)		P
Disable Communication to PID Loop	DisableCommunicationToPidLoop(*PID Loop*)		P
Disable Communication to Point	DisableCommunicationToPoint(*Point*)		P
Disable Event/Reaction Group	DisableEventReactionGroup(*E/R Group*)		P
Disable I/O Unit Causing Current Error	DisableIoUnitCausingCurrentError()		P
Disable Mistic PID Output	DisableMisticPidOutput(*Of PID Loop*)		P
Disable Mistic PID Output Tracking in Manual Mode	DisableMisticPidOutputTrackingInManualMode(*On PID Loop*)		P
Disable Mistic PID Setpoint Tracking in Manual Mode	DisableMisticPidSetpointTrackingInManualMode(*On PID Loop*)		P
Disable Scanning for All Events	DisableScanningForAllEvents(*On I/O Unit*)		P
Disable Scanning for Event	DisableScanningForEvent(*Event/Reaction*)		P
Disable Scanning of Event/Reaction Group	DisableScanningOfEventReactionGroup(*E/R Group*)		P
Divide		x / y	F
Down Timer Expired?	HasDownTimerExpired(*Down Timer*)		F
Enable Communication to All I/O Points	EnableCommunicationToAllIoPoints()		P
Enable Communication to All I/O Units	EnableCommunicationToAllIoUnits()		P
Enable Communication to Event/Reaction	EnableCommunicationToEventReaction(*Event/Reaction*)		P
Enable Communication to I/O Unit	EnableCommunicationToIoUnit(*I/O Unit*)		P
Enable Communication to Mistic PID Loop	EnableCommunicationToMisticPidLoop(*PID Loop*)		P
Enable Communication to PID Loop	EnableCommunicationToPidLoop(*PID Loop*)		P
Enable Communication to Point	EnableCommunicationToPoint(*Point*)		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Enable Event/Reaction Group	EnableEventReactionGroup(*E/R Group*)		P
Enable I/O Unit Causing Current Error	EnableIoUnitCausingCurrentError()		P
Enable Mistic PID Output	EnableMisticPidOutput(*On PID Loop*)		P
Enable Mistic PID Output Tracking in Manual Mode	EnableMisticPidOutputTrackingInManualMode (*On PID Loop*)		P
Enable Mistic PID Setpoint Tracking in Manual Mode	EnableMisticPidSetpointTrackingInManual Mode(*On PID Loop*)		P
Enable Scanning for All Events	EnableScanningForAllEvents(*On I/O Unit*)		P
Enable Scanning for Event	EnableScanningForEvent(*Event/Reaction*)		P
Enable Scanning of Event/Reaction Group	EnableScanningOfEventReactionGroup()		P
Equal to Numeric Table Element?		n == nt[0]	F
Equal?		x == y	F
Erase Files in Permanent Storage	EraseFilesInPermanentStorage()		F
Error on I/O Unit?	IsErrorOnIoUnit()		F
Error?	IsErrorPresent()		F
Event Occurred?	HasEventOccurred(*Event/Reaction*)		F
Event Occurring?	IsEventOccurring(*Event/Reaction*)		F
Event Scanning Disabled?	IsEventScanningDisabled(*Event/Reaction*)		F
Event Scanning Enabled?	IsEventScanningEnabled(*Event/Reaction*)		F
Event/Reaction Communication Enabled?	IsEventReactionCommEnabled(*Event/Reaction*)		F
Event/Reaction Group Communication Enabled?	IsEventReactionGroupEnabled(*E/R Group*)		F
Find Character in String	FindCharacterInString(*Find, Start at Index, Of String*)		F
Find Substring in String	FindSubstringInString(*Find, Start at Index, Of String*)		F
Float Valid?	IsFloatValid(*Float*)		F
Generate Checksum on String	GenerateChecksumOnString(*Start Value, On String*)		F
Generate Forward CCITT on String	GenerateForwardCcittOnString(*Start Value, On String*)		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Generate Forward CRC-16 on String	GenerateForwardCrc16OnString(*Start Value, On String*)		F
Generate N Pulses	GenerateNPulses(*On Time (Seconds), Off Time (Seconds), Number of Pulses, On Point*)		P
Generate Random Number	GenerateRandomNumber()		F
Generate Reverse CCITT on String	GenerateReverseCcittOnString(*Start Value, On String*)		F
Generate Reverse CRC-16 on String	GenerateReverseCrc16OnString(*Start Value, On String*)		F
Generate Reverse CRC-16 on Table (32 bit)	GenerateReverseCrc16OnTable32(*Start Value, Table, Starting Element, Number of Elements*)		F
Get & Clear All HDD Module Off-Latches	GetClearAllHddModuleOnLatches(*I/O Unit, Start Index, Put Result In*)		F
Get & Clear All HDD Module On-Latches	GetClearHddModuleCounter(*I/O Unit, Module Number, Point Number, Put Result In*)		F
Get & Clear Analog Filtered Value	GetClearAnalogFilteredValue(*From*)		F
Get & Clear Analog Maximum Value	GetClearAnalogMaxValue(*From*)		F
Get & Clear Analog Minimum Value	GetClearAnalogMinValue(*From*)		F
Get & Clear Analog Totalizer Value	GetClearAnalogTotalizerValue(*From*)		F
Get & Clear Counter	GetClearCounter(*From Point*)		F
Get & Clear Event Latches	GetClearEventLatches(*E/R Group*)		
Get & Clear HDD Module Counter	GetClearHddModuleCounters(*I/O Unit, Module Number, Start Table Index, Put Result In*)		F
Get & Clear HDD Module Counters	GetClearHddModuleOffLatches(*I/O Unit, Module Number, Put Result In*)		F
Get & Clear HDD Module Off-Latches	GetClearHddModuleOnLatches(*I/O Unit, Module Number, Put Result In*)		F
Get & Clear HDD Module On-Latches	GetAllHddModuleOffLatches(*I/O Unit, Start Index, Put Result In*)		F
Get & Clear Off-Latch	GetClearOffLatch(*From Point*)		F
Get & Clear On-Latch	GetClearOnLatch(*From Point*)		F
Get & Restart Off-Pulse Measurement	GetRestartOffPulseMeasurement(*From Point*)		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Get & Restart Off-Time Totalizer	GetRestartOffTimeTotalizer(*From Point*)		F
Get & Restart On-Pulse Measurement	GetRestartOnPulseMeasurement(*From Point*)		F
Get & Restart On-Time Totalizer	GetRestartOnTimeTotalizer(*From Point*)		F
Get & Restart Period	GetRestartPeriod(*From Point*)		F
Get All HDD Module Off-Latches	GetAllHddModuleOnLatches(*I/O Unit, Start Index, Put Result In*)		F
Get All HDD Module On-Latches	GetAllHddModuleStates(*I/O Unit, Start Index, Put Result In*)		F
Get All HDD Module States	GetHddModuleCounters(*I/O Unit, Module Number, Start Table Index, Put Result In*)		F
Get Analog Filtered Value	GetAnalogFilteredValue(*From*)		F
Get Analog Maximum Value	GetAnalogMaxValue(*From*)		F
Get Analog Minimum Value	GetAnalogMinValue(*From*)		F
Get Analog Square Root Filtered Value	GetAnalogSquareRootFilteredValue(*From*)		F
Get Analog Square Root Value	GetAnalogSquareRootValue(*From*)		F
Get Analog Totalizer Value	GetAnalogTotalizerValue(*From*)		F
Get Available File Space	GetAvailableFileSpace(*File System Type*)		F
Get Chart Status	GetChartStatus(*Chart*)		F
Get Communication Handle Value	GetCommunicationHandleValue(*From, To*)		P
Get Control Engine Address	GetControlEngineAddress()		F
Get Control Engine Type	GetEngineType()		F
Get Counter	GetCounter(*From Point*)		F
Get Day	GetDay()		F
Get Day of Week	GetDayOfWeek()		F
Get End-Of-Message Terminator	GetEndOfMessageTerminator(*Communication Handle*)		F
Get Error Code of Current Error	GetErrorCodeOfCurrentError()		F
Get Error Count	GetErrorCount()		F
Get Event Latches	GetEventLatches(*E/R Group*)		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Get Firmware Version	GetFirmwareVersion(*Put in*)		P
Get Frequency	GetFrequency(*From Point*)		F
Get HDD Module Counters	GetHddModuleOffLatches(*I/O Unit, Module Number, Put Result In*)		F
Get HDD Module Off-Latches	GetHddModuleOnLatches(*I/O Unit, Module Number, Put Result In*)		F
Get HDD Module On-Latches	GetHddModuleStates(*I/O Unit, Module Number, Put Result In*)		F
Get HDD Module States	GetClearAllHddModuleOffLatches(*I/O Unit, Start Index, Put Result In*)		F
Get High Bits of Integer 64	GetHighBitsOfInt64(*High Bits From*)		F
Get Hours	GetHours()		F
Get I/O Unit as Binary Value	GetIoUnitAsBinaryValue(*I/O Unit*)		F
Get I/O Unit Event Message State	GetIoUnitEventMsgState(*I/O Unit, Event Message #, Put Result in*);		F
Get I/O Unit Event Message Text	GetIoUnitEventMsgText(*I/O Unit, Event Message #, Put Result in*);		F
Get I/O Unit Scratch Pad Bits	GetIoUnitScratchPadBits(*I/O Unit, Put Result in*);		F
Get I/O Unit Scratch Pad Float Element	GetIoUnitScratchPadFloatElement(*I/O Unit, Index, Put Result in*);		F
Get I/O Unit Scratch Pad Float Table	GetIoUnitScratchPadFloatTable(*I/O Unit, Length, From Index, To Index, To Table*);		F
Get I/O Unit Scratch Pad Integer 32 Element	GetIoUnitScratchPadInt32Element(*I/O Unit, Index, Put Result in*);		F
Get I/O Unit Scratch Pad Integer 32 Table	GetIoUnitScratchPadInt32Table(*I/O Unit, Length, From Index, To Index, To Table*);		F
Get I/O Unit Scratch Pad String Element	GetIoUnitScratchPadStringElement(*I/O Unit, Index, Put Result in*);		F
Get I/O Unit Scratch Pad String Table	GetIoUnitScratchPadStringTable(*I/O Unit, Length, From Index, To Index, To Table*);		F
Get ID of Block Causing Current Error	GetIdOfBlockCausingCurrentError()		F
Get Julian Day	GetJulianDay()		F
Get Length of Table	GetLengthOfTable(*Table*)		F
Get Line Causing Current Error	GetLineCausingCurrentError()		F
Get Low Bits of Integer 64	GetLowBitsOfInt64(*Integer 64*)		F
Get Minutes	GetMinutes()		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Get Mistic PID Control Word	GetMisticPidControlWord(*From PID Loop*)		F
Get Mistic PID D Term	GetMisticPidDTerm(*From PID Loop*)		F
Get Mistic PID I Term	GetMisticPidITerm(*From PID Loop*)		F
Get Mistic PID Input	GetMisticPidInput(*PID Loop*)		F
Get Mistic PID Mode	GetMisticPidMode(*PID Loop*)		F
Get Mistic PID Output	GetMisticPidOutput(*PID Loop*)		F
Get Mistic PID Output Rate of Change	GetMisticPidOutputRateOfChange(*From PID Loop*)		F
Get Mistic PID P Term	GetMisticPidPTerm(*From PID Loop*)		F
Get Mistic PID Scan Rate	GetMisticPidScanRate(*From PID Loop*)		F
Get Mistic PID Setpoint	GetMisticPidSetpoint(*PID Loop*)		F
Get Month	GetMonth()		F
Get Name of Chart Causing Current Error	GetNameOfChartCausingCurrentError(*Put in*)		P
Get Name of I/O Unit Causing Current Error	GetNameOfIoUnitCausingCurrentError(*Put in*)		P
Get Nth Character	GetNthCharacter(*From String, Index*)	x = s[n];	F
Get Number of Characters Waiting	GetNumCharsWaiting(*On Communication Handle*)		F
Get Off-Latch		See Off-Latch Set?	F
Get Off-Pulse Measurement	GetOffPulseMeasurement(*From Point*)		F
Get Off-Pulse Measurement Complete Status	GetOffPulseMeasurementCompleteStatus(*From Point*)		F
Get Off-Time Totalizer	GetOffTimeTotalizer(*From Point*)		F
Get On-Latch	GetOnLatch(*On Point*)	See On-Latch Set?	F
Get On-Pulse Measurement	GetOnPulseMeasurement(*From Point*)		F
Get On-Pulse Measurement Complete Status	GetOnPulseMeasurementCompleteStatus(*From Point*)		F
Get On-Time Totalizer	GetOnTimeTotalizer(*From Point*)		F
Get Period	GetPeriod(*From Point*)		F
Get Period Measurement Complete Status	GetPeriodMeasurementCompleteStatus(*From Point*)		F
Get PID Configuration Flags	GetPidConfigFlags(*PID Loop*)		F
Get PID Current Input	GetPidCurrentInput(*PID Loop*)		F
Get PID Current Setpoint	GetPidCurrentSetpoint(*PID Loop*)		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Get PID Feed Forward	GetPidFeedForward(*PID Loop*)		F
Get PID Feed Forward Gain	GetPidFeedForwardGain(*PID Loop*)		F
Get PID Forced Output When Input Over Range	GetPidForcedOutputWhenInputOverRange(*PID Loop*)		F
Get PID Forced Output When Input Under Range	GetPidForcedOutputWhenInputUnderRange(*PID Loop*)		F
Get PID Gain	GetPidGain(*PID Loop*)		F
Get PID Input	GetPidInput(*PID Loop*)		F
Get PID Input High Range	GetPidInputHighRange(*PID Loop*)		F
Get PID Input Low Range	GetPidInputLowRange(*PID Loop*)		F
Get PID Max Output Change	GetPidMaxOutputChange(*PID Loop*)		F
Get PID Min Output Change	GetPidMinOutputChange(*PID Loop*)		F
Get PID Mode	GetPidMode(*PID Loop*)		F
Get PID Output	GetPidOutput(*PID Loop*)		F
Get PID Output High Clamp	GetPidOutputHighClamp(*PID Loop*)		F
Get PID Output Low Clamp	GetPidOutputLowClamp(*PID Loop*)		F
Get PID Scan Time	GetPidScanTime(*PID Loop*)		F
Get PID Setpoint	GetPidSetpoint(*PID Loop*)		F
Get PID Status Flags	GetPidStatusFlags(*PID Loop*)		F
Get PID Tune Derivative	GetPidTuneDerivative(*PID Loop*)		F
Get PID Tune Integral	GetPidTuneIntegral(*PID Loop*)		F
Get Pointer From Name	GetPointerFromName(*Name, Pointer*)		P
Get Seconds	GetSeconds()		F
Get Seconds Since Midnight	GetSecondsSinceMidnight()		F
Get Severity of Current Error	GetSeverityOfCurrentError()		F
Get String Length	GetStringLength(*Of String*)		F
Get Substring	GetSubstring(*From String, Start at Index, Num. Characters, Put Result in*)		P
Get System Time	GetSystemTime()		F
Get Target Address State	GetTargetAddressState(*Enable Mask, Active Mask, I/O Unit*)		P
Get Type From Name	GetTypeFromName(*Name*)		F
Get Value From Name	GetValueFromName(*Name*)		F
Get Year	GetYear()		F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Greater Than Numeric Table Element?		`x > nt[0]`	F
Greater Than or Equal to Numeric Table Element?		`x >= nt[0]`	F
Greater Than or Equal?		`x >= y`	F
Greater?		`x > y`	F
Hyperbolic Cosine	`HyperbolicCosine(Of)`		F
Hyperbolic Sine	`HyperbolicSine(Of)`		F
Hyperbolic Tangent	`HyperbolicTangent(Of)`		F
I/O Point Communication Enabled?	`IsIoPointCommEnabled(I/O Point)`		F
I/O Unit Communication Enabled?	`IsIoUnitCommEnabled(I/O Unit)`		F
I/O Unit Ready?	`IsIoUnitReady(I/O Unit)`		F
Increment Variable	`IncrementVariable(Variable)`	`x = x + 1;`	P
IVAL Move Numeric Table to I/O Unit	`IvalMoveNumTableToIoUnit(Start at Index, Of Table, Move to)`		P
IVAL Set Analog Point	`IvalSetAnalogPoint(To, On Point)`		P
IVAL Set Counter	`IvalSetCounter(To, On Point)`		P
IVAL Set Frequency	`IvalSetFrequency(To, On Point)`		P
IVAL Set I/O Unit from MOMO Masks	`IvalSetIoUnitFromMomo(On Mask, Off Mask, On I/O Unit)`		P
IVAL Set Mistic PID Control Word	`IvalSetMisticPidControlWord(On Mask, Off Mask, For PID Loop)`		P
IVAL Set Mistic PID Process Term	`IvalSetMisticPidProcessTerm(To, On PID Loop)`		P
IVAL Set Off-Latch	`IvalSetOffLatch(To, On Point)`		P
IVAL Set Off-Pulse	`IvalSetOffPulse(To, On Point)`		P
IVAL Set Off-Totalizer	`IvalSetOffTotalizer(To, On Point)`		P
IVAL Set On-Latch	`IvalSetOnLatch(To, On Point)`		P
IVAL Set On-Pulse	`IvalSetOnPulse(To, On Point)`		P
IVAL Set On-Totalizer	`IvalSetOnTotalizer(To, On Point)`		P
IVAL Set Period	`IvalSetPeriod(To, On Point)`		P
IVAL Set TPO Percent	`IvalSetTpoPercent(To, On Point)`		P
IVAL Set TPO Period	`IvalSetTpoPeriod(Value, On Point)`		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
IVAL Turn Off	`IvalTurnOff(`*`Point`*`)`		P
IVAL Turn On	`IvalTurnOn(`*`Point`*`)`		P
Less Than Numeric Table Element?		`x < nt[0]`	F
Less Than or Equal to Numeric Table Element?		`x <= nt[0]`	F
Less Than or Equal?		`x <= y`	F
Less?		`x < y`	F
Listen for Incoming Communication	`ListenForIncomingCommunication (`*`Communication Handle`*`);`		F
Load Files From Permanent Storage	`LoadFilesFromPermanentStorage`		F
Make Integer 64	`MakeInt64(`*`High Integer, Low Integer`*`)`		F
Maximum	`Max(`*`Compare, With`*`)`		F
Minimum	`Min(`*`Compare, With`*`)`		F
Modulo		`x % y`	F
Move		`x = y;`	P
Move 32 Bits	`Move32Bits(`*`From, To`*`)`		P
Move from Numeric Table Element		`x = nt[0];`	F
Move from Pointer Table Element		`pn = pt[0];`	F
Move from String Table Element		`s = st[0];`	P
Move I/O Unit to Numeric Table	`MoveIoUnitToNumTable(`*`I/O Unit, Starting Index, Of Table`*`)`		P
Move Numeric Table Element to Numeric Table		`nt1[0] = nt2[5];`	P
Move Numeric Table to I/O Unit	`MoveNumTableToIoUnit(`*`Start at Index, Of Table, Move to`*`)`		P
Move Numeric Table to Numeric Table	`MoveNumTableToNumTable(`*`From Table, From Index, To Table, To Index, Length`*`)`		P
Move String		`s1 = s2;`	P
Move to Numeric Table Element		`nt[0] = x;`	P
Move to Numeric Table Elements	`MoveToNumTableElements(`*`From, Start Index, End Index, Of Table`*`)`		P
Move to Pointer		`pn = &n;`	F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Move to Pointer Table Element		pt[0] = &n;	F
Move to String Table Element		st[0] = s;	P
Move to String Table Elements	MoveToStrTableElements(*From, Start Index, End Index, Of Table*)		P
Multiply		x * y	F
Natural Log	NaturalLog(*Of*)		F
NOT		not x	F
Not Equal to Numeric Table Element?		n <> nt[0]	F
Not Equal?		x <> y	F
NOT?		not x	F
Numeric Table Element Bit Clear	NumTableElementBitClear(*Element Index, Of Integer Table, Bit to Clear*)		P
Numeric Table Element Bit Set	NumTableElementBitSet(*Element Index, Of Integer Table, Bit to Set*)		P
Numeric Table Element Bit Test	NumTableElementBitTest(*Element Index, Of Integer Table, Bit to Test*)		F
Off?	IsOff(*Point*)	di == 0	F
Off-Latch Set?	IsOffLatchSet(*On Point*)		F
On?	IsOn(*Point*)	di == 1	F
On-Latch Set?	IsOnLatchSet(*On Point*)		F
Open Outgoing Communication	OpenOutgoingCommunication(*Communication Handle*)		F
OR		x or y	F
OR?		See OR	F
Pause Timer	PauseTimer(*Timer*)		P
PID Loop Communication Enabled?	IsPidLoopCommEnabled(*PID Loop*)		F
Pointer Equal to Null?		pn == null	F
Pointer Table Element Equal to Null?		pt[0] == null	F
Raise e to Power	RaiseEToPower(*Exponent*)		F
Raise to Power	Power(*Raise, To the*)		F
Ramp Analog Output	RampAnalogOutput(*Ramp Endpoint, Units/Sec, Point to Ramp*)		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Read Event/Reaction Hold Buffer	ReadEventReactionHoldBuffer (*Event/Reaction*)		F
Read Number from I/O Unit Memory Map	ReadNumFromIoUnitMemMap(*I/O Unit, Mem address, To*)		F
Read Numeric Table from I/O Unit Memory Map	ReadNumTableFromIoUnitMemMap(*Length, Start Index, I/O Unit, Mem address, To*)		F
Read String from I/O Unit Memory Map	ReadStrFromIoUnitMemMap(*Length, I/O Unit, Mem address, To*)		F
Read String Table from I/O Unit Memory Map	ReadStrTableFromIoUnitMemMap(*Length, Start Index, I/O Unit, Mem address, To*)		F
Receive Character	ReceiveChar(*Communication Handle*)		F
Receive N Characters	ReceiveNChars(*Put In, Number of Characters, Communication Handle*)		F
Receive Numeric Table	ReceiveNumTable(*Length, Start at Index, Of Table, Communication Handle*)		F
Receive String	ReceiveString(*Put In, Communication Handle*)		F
Remove Current Error and Point to Next Error	RemoveCurrentError()		P
Retrieve Strategy CRC	RetrieveStrategyCrc()		F
Round	Round(*Value*)		F
Save Files To Permanent Storage	SaveFilesToPermanentStorage()		F
Seed Random Number	SeedRandomNumber()		P
Send Communication Handle Command	SendCommunicationHandleCommand (*Communication Handle, Command*)		F
Set All Target Address States	SetAllTargetAddressStates(*Must-On Mask, Must-Off Mask, Active Mask*)		P
Set Analog Filter Weight	SetAnalogFilterWeight(*To, On Point*)		P
Set Analog Gain	SetAnalogGain(*To, On Point*)		P
Set Analog Load Cell Fast Settle Level	SetAnalogLoadCellFastSettleLevel(*To, On Point*)		P
Set Analog Load Cell Filter Weight	SetAnalogLoadCellFilterWeight(*To, On Point*)		P
Set Analog Offset	SetAnalogOffset(*To, On Point*)		P
Set Analog Totalizer Rate	SetAnalogTotalizerRate(*To Seconds, On Point*)		P
Set Analog TPO Period	SetAnalogTpoPeriod(*To, On Point*)		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Set Communication Handle Value	SetCommunicationHandleValue(*From, To*)		P
Set Date	SetDate(*To*)		P
Set Day	SetDay(*To*)		P
Set Down Timer Preset Value	SetDownTimerPreset(*Target Value, Down Timer*)		P
Set End-Of-Message Terminator	SetEndOfMessageTerminator(*Communication Handle, To Character*)		P
Set HDD Module from MOMO Masks	SetHddModulefromMOMOMasks(*I/O Unit, Module Number, Must-On Mask, Must-Off Mask*)		F
Set Hours	SetHours(*To*)		P
Set I/O Unit Event Message State	SetIoUnitEventMessageState(*I/O Unit, Event Message #, State*);		F
Set I/O Unit Event Message Text	SetIoUnitEventMessageText(*I/O Unit, Event Message #, Message Text*);		F
Set I/O Unit from MOMO Masks	SetIoUnitFromMomo(*Must-On Mask, Must-Off Mask, Digital I/O Unit*)		P
Set I/O Unit Scratch Pad Bits from MOMO Mask	SetIoUnitScratchPadBitsFromMomoMask(*I/O Unit, Must-on Mask, Must-off Mask*);		F
Set I/O Unit Scratch Pad Float Element	SetIoUnitScratchPadFloatElement(*I/O Unit, Index, From*);		F
Set I/O Unit Scratch Pad Float Table	SetIoUnitScratchPadFloatTable(*I/O Unit, Length, To Index, From Index, From Table*);		F
Set I/O Unit Scratch Pad Integer 32 Element	SetIoUnitScratchPadInt32Element(*I/O Unit, Index, From*);		F
Set I/O Unit Scratch Pad Integer 32 Table	SetIoUnitScratchPadInt32Table(*I/O Unit, Length, To Index, From Index, From Table*);		F
Set I/O Unit Scratch Pad String Element	SetIoUnitScratchPadStringElement(*I/O Unit, Index, From*);		F
Set I/O Unit Scratch Pad String Table	SetIoUnitScratchPadStringTable(*I/O Unit, Length, To Index, From Index, From Table*);		F
Set Minutes	SetMinutes(*To*)		P
Set Mistic PID Control Word	SetMisticPidControlWord(On-*Mask, Off-Mask, For PID Loop*)		P
Set Mistic PID D Term	SetMisticPidDTerm(*To, On PID Loop*)		P
Set Mistic PID I Term	SetMisticPidITerm(*To, On PID Loop*)		P
Set Mistic PID Input	SetMisticPidInput(*PID Loop, Input*)		P
Set Mistic PID Mode to Auto	SetMisticPidModeToAuto(*On PID Loop*)		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Set Mistic PID Mode to Manual	`SetMisticPidModeToManual(`*`On PID Loop`*`)`		P
Set Mistic PID Output Rate of Change	`SetMisticPidOutputRateOfChange(`*`To, On PID Loop`*`)`		P
Set Mistic PID P Term	`SetMisticPidPTerm(`*`To, On PID Loop`*`)`		P
Set Mistic PID Scan Rate	`SetMisticPidScanRate(`*`To, On PID Loop`*`)`		P
Set Mistic PID Setpoint	`SetMisticPidSetpoint(`*`PID Loop, Setpoint`*`)`		P
Set Month	`SetMonth(`*`To`*`)`		P
Set Nth Character	`SetNthCharacter(`*`To, In String, At Index`*`)`	`s[n] = x;`	F
Set PID Configuration Flags	`SetPidConfigFlags(`*`PID Loop, Configuration Flags`*`)`		P
Set PID Feed Forward	`SetPidFeedForward(`*`PID Loop, Feed Forward`*`)`		P
Set PID Feed Forward Gain	`SetPidFeedForwardGain(`*`PID Loop, Feed Fwd Gain`*`)`		P
Set PID Forced Output When Input Over Range	`SetPidForcedOutputWhenInputOverRange(`*`PID Loop, Forced Output`*`)`		P
Set PID Forced Output When Input Under Range	`SetPidForcedOutputWhenInputUnderRange(`*`PID Loop, Forced Output`*`)`		P
Set PID Gain	`SetPidGain(`*`PID Loop, Gain`*`)`		P
Set PID Input	`SetPidInput(`*`PID Loop, Input`*`)`		P
Set PID Input High Range	`SetPidInputHighRange(`*`PID Loop, High Range`*`)`		P
Set PID Input Low Range	`SetPidInputLowRange(`*`PID Loop, Low Range`*`)`		P
Set PID Max Output Change	`SetPidMaxOutputChange(`*`PID Loop, Max Change`*`)`		P
Set PID Min Output Change	`SetPidMinOutputChange(`*`PID Loop, Min Change`*`)`		P
Set PID Mode	`SetPidMode(`*`PID Loop, Mode`*`)`		P
Set PID Output	`SetPidOutput(`*`PID Loop, Output`*`)`		P
Set PID Output High Clamp	`SetPidOutputHighClamp(`*`PID Loop, High Clamp`*`)`		P
Set PID Output Low Clamp	`SetPidOutputLowClamp(`*`PID Loop, Low Clamp`*`)`		P
Set PID Scan Time	`SetPidScanTime(`*`PID Loop, Scan Time`*`)`		P
Set PID Setpoint	`SetPidSetpoint(`*`PID Loop, Setpoint`*`)`		P
Set PID Tune Derivative	`SetPidTuneDerivative(`*`PID Loop, Derivative`*`)`		P
Set PID Tune Integral	`SetPidTuneIntegral(`*`PID Loop, Integral`*`)`		P
Set Seconds	`SetSeconds(`*`To`*`)`		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Set Target Address State	SetTargetAddressState(*Must-On Mask, Must-Off Mask, Active Mask, I/O Unit*)		P
Set Time	SetTime(*To*)		P
Set TPO Percent	SetTpoPercent(*To Percent, On Point*)		P
Set TPO Period	SetTpoPeriod(*To Seconds, On Point*)		P
Set Up Timer Target Value	SetUpTimerTarget(*Target Value, Up Timer*)		P
Set Variable False	SetVariableFalse(*Variable*)		P
Set Variable True	SetVariableTrue(*Variable*)		P
Set Year	SetYear(*To*)		P
Shift Numeric Table Elements	ShiftNumTableElements(*Shift Count, Table*)		P
Sine	Sine(*Of*)		F
Square Root	SquareRoot(*Of*)		F
Start Continuous Square Wave	StartContinuousSquareWave(*On Time (Seconds), Off Time (Seconds), On Point*)		P
Start Counter	StartCounter(*On Point*)		P
Start Off-Pulse	StartOffPulse(*Off Time (Seconds), On Point*)		P
Start On-Pulse	StartOnPulse(*On Time (Seconds), On Point*)		P
Start Timer	StartTimer(*Timer*)		P
Stop Chart on Error	StopChartOnError()		P
Stop Counter	StopCounter(*On Point*)		P
Stop Timer	StopTimer(*Timer*)		P
String Equal to String Table Element?		`s == st[0]`	F
String Equal?		`s1 == s2`	F
Subtract		`x - y`	F
Suspend Chart on Error	SuspendChartOnError()		F
Tangent	Tangent(*Of*)		F
Test Equal		See Equal?	F
Test Equal Strings		See String Equal?	F
Test Greater		See Greater?	F
Test Greater or Equal		See Greater Than or Equal?	F
Test Less		See Less?	F
Test Less or Equal		See Less Than or Equal?	F

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Test Not Equal		See Not Equal?	F
Test Within Limits		See Within Limits?	F
Timer Expired?	HasTimerExpired(*Timer*)		F
Transfer N Characters	TransferNChars(*Destination Handle, Source Handle, Num Chars*)		F
Transmit Character	TransmitChar(*Character, Communication Handle*)		F
Transmit NewLine	TransmitNewLine(*Communication Handle*)		F
Transmit Numeric Table	TransmitNumTable(*Length, Start at Index, Of Table, Communication Handle*)		F
Transmit String	TransmitString(*String, Communication Handle*)		F
Transmit/Receive String	TransmitReceiveString(*String, Communication Handle, Put Result in*)		F
Truncate	Truncate(*Value*)		F
Turn Off	TurnOff(*Output*)	doOutput = 0;	P
Turn Off HDD Module Point	TurnOffHDDModulePoint(*I/O Unit, Module Number, Point Number*)		F
Turn On	TurnOn(*Output*)	doOutput = 1;	P
Turn On HDD Module Point	TurnOnHddModulePoint(*I/O Unit, Module Number, Point Number*)		F
Up Timer Target Time Reached?	HasUpTimerReachedTargetTime(*Up Timer*)		F
Variable False?	IsVariableFalse(*Variable*)	not x	F
Variable True?	IsVariableTrue(*Variable*)	x	F
Verify Checksum on String	VerifyChecksumOnString(*Start Value, On String*)		F
Verify Forward CCITT on String	VerifyForwardCcittOnString(*Start Value, On String*)		F
Verify Forward CRC-16 on String	VerifyForwardCrc16OnString(*Start Value, On String*)		F
Verify Reverse CCITT on String	VerifyReverseCcittOnString(*Start Value, On String*)		F
Verify Reverse CRC-16 on String	VerifyReverseCrc16OnString(*Start Value, On String*)		F
Within Limits?	IsWithinLimits(*Value, Low Limit, High Limit*)	(x >= nLoLimit) and (x <= nHiLimit)	F
Write I/O Unit Configuration to EEPROM	WriteIoUnitConfigToEeprom(*On I/O Unit*)		P

Action/Condition Command Name	OptoScript Command (with Arguments)	OptoScript Equivalent Example	Type
Write Number to I/O Unit Memory Map	`WriteNumToIoUnitMemMap(`*`I/O Unit, Mem address, Variable`*`)`		F
Write Numeric Table to I/O Unit Memory Map	`WriteNumTableToIoUnitMemMap(`*`Length, Start Index, I/O Unit, Mem address, Table`*`)`		F
Write String Table to I/O Unit Memory Map	`WriteStrTableToIoUnitMemMap(`*`Length, Start Index, I/O Unit, Mem address, Table`*`)`		F
Write String to I/O Unit Memory Map	`WriteStrToIoUnitMemMap(`*`I/O Unit, Mem address, Variable`*`)`		F
XOR		`x xor y`	F
XOR?		See XOR	F

OptoScript Language Reference

Introduction

This appendix includes the following reference information about the OptoScript language:

OptoScript Comparison with Standard Programming Languages

The tables on the following pages compare OptoScript functions and variables to those available in Pascal, BASIC, and C. For more information on using OptoScript, see Chapter 11 and Appendix E.

General Notes:

1. The BASIC column is based on Microsoft's Visual Basic language.
2. The Pascal column is based on Borland's ObjectPascal language.
3. The use of logical statements in BASIC and Pascal is significantly different than in OptoScript and C. BASIC and Pascal have a Boolean data type; OptoScript and C use integers. OptoScript and C treat a zero value as false and a non-zero value as true.
4. In OptoScript, you cannot use a break type of command in a loop.
5. OptoScript can test only one case in a switch at a time; other languages can test more than one.

Function Comparison

	OptoScript	BASIC[1]	C	Pascal[2]
integer 32 (decimal)	123	123	123	123
integer 32 (hexadecimal)	0x123ABC	&H123ABC	0x123ABC	&123ABC
integer 64 (decimal)	123i64	Not available	123LL or 123i64	123
integer 64 (hexadecimal)	0x123ABCi64	Not available	0x123ABCLL or 0x123ABCi64	&123ABC
float (standard)	12.34	12.34	12.34	12.34
float (scientific)	12.34E+17	1.234E+18	1.234E+17	12.34E+17
string	"hello"	"hello"	"hello"	'hello'
character	65 'A'	Not available	65 'A'	'A'
block comment	/* */	Not available	/* */	{ }
line comment	//	'	//	//
numeric assignment	n = 3;	n = 3	n = 3;	n := 3;

numeric table assignment	t[0] = 1;
	t[i] = 2;
	t[i+1] = 3;
		t(0) = 1
		t(i) = 2
		t(i + 1) = 3
			t[0] = 1;
			t[i] = 2;
			t[i+1] = 3;
				t[0] := 1;
				t[i] := 2;
				t[i + 1] := 3;

numeric expressions	i = (f * 2.0) + t[3];
	t[4] = n + ((x – y) * z);
		i = (f * 2.0) + t(3)
		t(4) = n + ((x – y) * z)
			i = (f * 2.0) + t[3];
			t[4] = n + ((x – y) * z);
				i := (f * 2.0) + t[3];
				t[4] := n + ((x – y) * z);

string assignment	s = "hello";
	s = s2;
		s = "hello"
		s = s2
			strcpy(s, "hello");
			strcpy(s, s2);
				s := 'hello';
				s := s2;

string table assignment	st[0] = s;
	st[1] = "hello";
	st[1+i] = st[5];
		st(0) = s
		st(1) = "hello"
		st(1 + i) = st(5)
			strcpy(st[0], s);
			strcpy(st[1],"hello");
			strcpy(st[1+i], st[5]);
				st[0] := s;
				st[1] := 'hello';
				st[1+i] := st[5];

string characters	n = s[0];
	s[0] = 'A';
		Not available	n = s[0];
			s[0] = 'A';
				s[0] := 'A';

	OptoScript	BASIC[1]	C	Pascal[2]

string expressions	`s = "hello" + s2 + s3;`
	`s = "hello" + Chr(n);`
		`s = "hello" + s2 + s3;`
		`s = "hello" + Chr(n)`
			`strcpy(s, "hello");`
			`strcat(s, s2);`
			`sprintf(s, "hello%c", n);`
				`s := 'hello' + s2 + s3;`
				`s := 'hello' + Chr(n);`

equal	`x == y`	`x = y`	`x == y`	`x = y`
not equal	`x <> y`	`x <> y`	`x != y`	`x <> y`
less than	`x < y`	`x < y`	`x < y`	`x < y`
less than or equal	`x <= y`	`x <= y`	`x <= y`	`x <= y`
greater than	`x > y`	`x > y`	`x > y`	`x > y`
greater than or equal	`x >= y`	`x <= y`	`x >= y`	`x <= y`

logical OR (See Note 3)	`x or y`	`(x <> 0) Or (y <> 0)`
		`a Or b  (booleans only)`
			`x || y`	`(x <> 0) Or (y <> 0)`
				`a Or b  (booleans only)`

logical AND	`x and y`	`(x <> 0) And (y <> 0)`
		`x And y  (booleans only)`
			`x && y`	`(x <> 0) And (y <> 0)`
				`a And b  (booleans only)`

logical XOR	`x xor y`	`(x <> 0) Xor (y <> 0)`
		`x Xor y  (booleans only)`
			`Not available`	`(x <> 0) Xor (y <> 0)`
				`a Xor b  (booleans only)`

logical NOT	`not x`	`Not (x <> 0)`
		`Not b (booleans only)`
			`!x`	`Not (x <> 0)`
				`Not b (booleans only)`

logical expressions	`(x >= 0) and (y == 3)`
	`not ((x>0)or(not y==3))`
		`(x >= 0) And (y = 3)`
		`Not((x > 0) Or ( y <> 3))`
			`(x >= 0) && (y == 3)`
			`!((x > 0) || (y != 3)`
				`(x >= 0) and (y = 3)`
				`not((x > 0) or ( y <> 3))`

bitwise NOT	`bitnot x`	`Not x`	`~x`	`not x`
bitwise OR (See Note 3)	`x bitor y`	`x Or y`	`x | y`	`x or y`
bitwise AND	`x bitand y`	`x And y`	`x & y`	`x and y`
bitwise XOR	`x bitxor y`	`x Xor y`	`x ^ y`	`x xor y`
bitwise shift left	`x << y`	`Not available`	`x << y`	`x shl y`
bitwise shift right	`x >> y`	`Not available`	`x >> y`	`x shr y`

	OptoScript	BASIC[1]	C	Pascal[2]

bitwise expressions	`i = x bitand`
	`0x0000FFFF;`
	`i = x << (i * 2);`
		`i = x And &H0000FFFF`
		`Not available`
			`i = x & 0x0000FFFF;`
			`i = x << (i * 2);`
				`i := x and &0000FFFF;`
				`i := x shl (i * 2);`

if statement	`if (i == 2) then`
	`  // do something`
	`endif`
		`If (i == 2) Then`
		`  // do something`
		`End If`
			`if (i == 2)`
			`{`
			`  // do something`
			`}`
				`if (i = 2) then`
				`begin`
				`  // do something`
				`end`

if/else statement	`if (i == 2) then`
	`  // do something`
	`else`
	`  // do something else`
	`endif`
		`If (i == 2) Then`
		`  // do something`
		`Else`
		`  // do something else`
		`End If`
			`if (i == 2)`
			`{`
			`  // do something`
			`}`
			`else`
			`{`
			`  // do something else`
			`}`
				`if (i = 2) then`
				`  begin`
				`    // do something`
				`  end`
				`else`
				`  begin`
				`    // do something else`
				`  end`

if/else if statement	`if (i == 2) then`
	`  // do something`
	`elseif (i >= 5) then`
	`  // do something else`
	`else`
	`  // do something else`
	`endif`
		`If (i = 2)`
		`  // do something`
		`ElseIf (i >= 5) Then`
		`  // do something else`
		`Else`
		`  // do something else`
		`End If`
			`if (i == 2)`
			`{`
			`  // do something`
			`}`
			`else if (i >= 5)`
			`{`
			`  // do something else`
			`}`
			`else`
			`{`
			`  // do something else`
			`}`
				`if (i = 2)`
				`  begin`
				`    // do something`
				`  end`
				`else if (i >= 5)`
				`  begin`
				`    // do something else`
				`  end`
				`else`
				`  begin`
				`    // do something else`
				`  end`

for loop (See Note 4)	`for i = 0 to 5 step 1`
	`  MyTable[i] = i * 2;`
	`next`
		`For i = 0 To 5 Step 1`
		`  MyTable(i) = i * 2`
		`Next`
			`for (i = 0; i < 5 ; i++)`
			`{`
			`  MyTable[i] = i * 2;`
			`}`
				`for i := 0 to 5 do`
				`begin`
				`  MyTable[i] := i * 2;`
				`end`

while loop (See Note 4)	`while (i < 5)`
	`  MyTable[i] = i * 2;`
	`  i = i + 1;`
	`wend`
		`While (i < 5)`
		`  MyTable(i) = i * 2`
		`  i = i + 1`
		`Wend`
			`while (i < 5)`
			`{`
			`  MyTable[i] = i * 2;`
			`  i = i + 1;`
			`}`
				`while (i < 5) do`
				`begin`
				`  MyTable[i] := i * 2;`
				`  i := i + 1;`
				`end`

	OptoScript	BASIC[1]	C	Pascal[2]

repeat loop (See Note 4)	`repeat`
	`  MyTable[i] = i * 2;`
	`  i = i + 1;`
	`until (i > 5);`
		`Do`
		`  MyTable(i) = i * 2`
		`  i = i + 1`
		`Loop Until (i > 5)`
			`do`
			`{`
			`  MyTable[i] = i * 2;`
			`  i = i + 1;`
			`} while !(i > 5);`
				`repeat`
				`  MyTable[i] := i * 2;`
				`  i = i + 1;`
				`until (i > 5);`

case (See Note 5)	`switch (i)`
	`  case 1:`
	`    f = 2.0 * x;`
	`    break`
	`  case z:`
	`    f = 2.0 * y;`
	`    break`
	`  default:`
	`    f = 2.0 * z;`
	`    break`
	`endswitch`
		`Select Case (i)`
		`  Case 1`
		`    f = 2.0 * x`
		`  Case z`
		`    f = 2.0 * y`
		`  Case Else`
		`    f = 2.0 * z;`
		`End Select`
			`switch (i)`
			`{`
			`  case 1:`
			`    f = 2.0 * x;`
			`    break;`
			`//case z:  NOT ALLOWED IN C`
			`//    f = 2.0 * y;`
			`//    break;`
			`  default:`
			`    f = 2.0 * z;`
			`    break;`
			`}`
				`case i of`
				`  1:`
				`    f := 2.0 * x;`
				`  2:`
				`    f := 2.0 * y;`
				`  else`
				`    f := 2.0 * z;`
				`end;`

Notes:

1. Based on Microsoft's Visual Basic language.
2. Based on Borland's ObjectPascal language.
3. The use of logical statements in BASIC and Pascal is significantly different than in OptoScript and C. BASIC and Pascal have a Boolean data type; OptoScript and C use integers. OptoScript and C treat a zero value as false and a non-zero value as true.
4. OptoScript cannot have a break type of command in a loop as is common with other lanuages.
5. OptoScript can test only one case at a time.

Variable Comparison

Variable Name	ioControl Type	BASIC Example	C Example	Pascal Example
n	integer 32	Dim n as Long	long n;	n: Integer;
nn	integer 64	Not available	LONGLONG d;	d: Int64;
f	float	Dim f as Single	float f;	f: Single;
s	string	Dim as String	char s[128];	s: ShortString;
p	pointer	not available	long * pn;	pn: ^Integer;
nt	integer 32 table	Dim nt(10) as Long	long i[10];	nt: array[0..9] of Integer;
ft	float 32 table	Dim ft(10) as Single	float f[10];	ft: array[0..9] of Single
st	string table	Dim st(10) as String	char s[10][128];	st: array[0..9] of ShortString;
pt	pointer table	not available	void * pt[10];	pt: array[0..9] of ^Integer;

Notes to Experienced Programmers

Experienced programmers, especially those who are new to ioControl, may be interested in the following notes.

Variable Database and Other Surprises

ioControl maintains a database of all declared variables—a notable difference from common procedural languages. Variables are not declared in the programming code, but in the ioControl tag database. This is a basic concept of ioControl and how it ties in with ioDisplay, but may seem odd to experienced programmers using ioControl for the first time. Also, all variables and objects are global. Local variables do not exist in ioControl in the way they do in most procedural languages. Subroutines in ioControl contain "local" variables, but those local variables apply throughout that subroutine.

Most languages allow you to return from a function before it ends, but OptoScript does not. The same effect can be achieved in other ways, however, such as introducing tests into the code. (Some people argue that this limitation produces better programming, because each function has only one exit point.)

ioControl's Target Audience

Because ioControl is based on OptoControl, which was conceived as a simple programming tool for non-programmers, it is designed to be relatively foolproof. Even though OptoScript provides advanced functionality, this philosophy also influenced the design of OptoScript. OptoScript exists only inside OptoScript blocks, which can only exist inside a flowchart. Flowcharts are the basis of ioControl.

Even an experienced programmer may want to think twice before using OptoScript blocks extensively. Many programmers like ioControl's simplicity not for themselves but for the field technicians and maintenance personnel who will use it in the field. While you could write an entire chart (or conceivably an entire strategy) in one block, doing so would eliminate most of the advantages of using ioControl. Consider limiting your use of OptoScript to math operations, complex string manipulation, and other logic most suited to scripting, so you retain ioControl's advantages for non-programmers.

Language Syntax

In the same way, OptoScript syntax is meant to be simple enough for a beginner to understand but also easy for an experienced programmer to learn quickly.

Some programmers may wonder why OptoScript is not modeled after just one existing language, such as BASIC or C. Instead, one can clearly see influences from Pascal, BASIC, and C. ioControl's target audience is one reason; internal consistency with ioControl commands and the capabilities and limitations of Opto 22 control engines are another.

Some aspects of OptoScript were designed to be consistent with ioControl commands. For instance, the *bitwise and* operator was named *bitand* in OptoScript because there is a command in ioControl named *Bit AND*.

OptoScript provides all the functionality of Opto 22 control engines but is subject to their limitations. For instance, OptoScript provides some convenient ways of working with strings, but only to a certain point.

For example, in an assignment statement, strings can be added together like this:

```
strDate = strMonth + "/" + strDay + "/" + strYear;
```

It would certainly be nice to use the same kind of string addition in a procedure call:

```
TransmitString(strMonth + "/" + strDay + "/" + strYear, nPort);
```

However, due to the current abilities of control engines, this type of string addition inside a function call is not possible.

OptoScript Lexical Reference

Token Syntax Legend

Tokens are the smallest units that the OptoScript compiler processes.

bold character:	specific character	*:	any character
(parenthesis):	content is treated as a unit	*letter*:	any character a through z, upper or lower case
[brackets]:	set of possible items		
opt subscript:	item is optional	*digit*:	one of [0 1 2 3 4 5 6 7 8 9]
no subscript:	item is not allowed	*non-zero-digit*	one of [1 2 3 4 5 6 7 8 9]
0+ subscript:	0 or more items may be chosen	*hex-digit*:	one of [0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f]
1 subscript:	one item must be chosen		
1+ subscript:	one or more items must be chosen		

Literals and Names

Token Name	Token Syntax	Comments
Int32Literal	`0[xX]1hex-digit1+`	Hexadecimal Integer 32-bit Good examples: `0x12AB, 0X12aB, 0x0` Bad examples: `x123ABC, 0123ABC`

	`0`
	`non-zero-digit1 digit0+`
		Decimal Integer 32-bit
		Good examples: `0, 123, 7890`
		Bad examples: `123ABC`

	`'[*'no]1'`	Single Character Good examples: `'A', '1', ' '` Bad examples: `'''`
Int64Literal	`0[xX]1hex-digit1+L`	Hexadecimal Integer 64 Good examples: `0x12ABL, 0X12aBL, 0x0L` Bad examples: `x123ABCL, 0123ABCL`

	`0`
	`digit1`
	`non-zero-digit0+L`
		Decimal Integer 64
		Good examples: `0L, 123L, 7890L`
		Bad examples: `123ABCL`

FloatLiteral	`digit0+ . digit1+ ([Ee]1 [+-]opt digit1+)opt`	Float Literal Good examples: `1.0, 2.3, 1.2e6, 1.2e-6, .1, .2e6` Bad examples: `1., 1.e7, 1e7`
StringLiteral	`"[*"no]0+"`	String Literal. Confined to single line. Good examples: `"abc def"` Bad examples: `"abc"def"`

Token Name	Token Syntax	Comments
NumericVariable StringVariable ChartVariable DigIoUnitVariable MixedIoUnitVariable PointerVariable NumericTable StringTable PointerTable CommunicationHandle	`[letter]1 [letter digit _ ]0+`	A letter followed by mix of letters, digits, and underscores. The name must be found in the ioControl database. Good examples: `MyInt, MyInt2, My_Int_3` Bad examples: `_MyInt, 0MyInt`
CommandProcedure CommandProcedureNoArgs CommandFunction CommandFunctionNoArgs	`[letter]1 [letter digit _ ]0+`	A letter followed by mix of letters, digits, and underscores. The name must be a built-in command or subroutine. Good examples: `Sine, Sine2, Sine_3` Bad examples: `_Sine, 0Sine`

Keywords (Reserved Words)

if
then
else
elseif
endif

for
to
step
next

switch
endswitch
case
break
default

while
do
wend

repeat
until

Chr

null

Operators

The following table lists operators in order of highest to lowest precedence:

Operator	Name/Meaning	Comments
-	negation	
`not`	logical not	
`bitnot`	bitwise not	
*	multiplication	
/	division	
%	modulo division	
-	subtraction	
+	addition	
+=	string append assignment	
<<	bitwise left shift	

Operator	Name/Meaning	Comments
>>	bitwise right shift	
==	equality	
<>	non-equality	
<	less than	
<=	less than or equal to	
>	greater than	
>=	greater than or equal to	
`bitand`	bitwise and	
`bitor`	bitwise or	
`bitxor`	bitwise exclusive or	
`and`	logical and	
`or`	logical or	
`xor`	logical exclusive or	
`not`	logical not	
`(  )`	parentheses	no precedence
`[  ]`	brackets	no precedence
`:`	colon	no precedence
`;`	semi-colon	no precedence
`,`	comma separator	no precedence
=	assignment	no precedence
&	address of	no precedence

Comments

OptoScript has two kinds of comments: single line and block.

Single line comments are indicated by two slashes, followed by any sequence of characters, until the end of the line.

Examples:

```
i = a + b; // this is a comment
i = a + b; //determine i by adding a and b together
// i = a + b; // This whole line is commented out
```

Block comments are indicated by a slash and an asterisk (`/*`), followed by any sequence of characters, and ending with an asterisk and a slash (`*/`). This type of comment may span multiple lines. Block comments may not be nested.

Examples:

```
i = a + b; /*determine i by adding a and b together*/
i = a + b; /* determine i by adding
a and b together */
/* i = a + b; // determine i by adding a and b together */
```

OptoScript Grammar Syntax Reference

Tokens are in regular type.
Keywords and operators are in **bold** type.
Syntax Rules are in *italic* type.

Program
→ *StatementList*

StatementList
→ *Statement*
→ *StatementList Statement*

Statement
→ *AssignmentStatement*
→ *StrAssignmentStatement*
→ *PtrAssignmentStatement*
→ *ProcedureCommand*
→ *FunctionCommand* **;**
→ *ConditionStatement*
→ *ForStatement*
→ *WhileStatement*
→ *RepeatStatement*
→ *SwitchStatement*

ProcedureCommand
→ CommandProcedureNoArgs **( ) ;**
→ CommandProcedure **(** *ArgumentList* **) ;**

FunctionCommand
→ CommandFunctionNoArgs **( )**
→ CommandFunction **(** *ArgumentList* **)**

ArgumentList
→ *NumericExp*
→ *ArgumentList* **,** *NumericExp*
→ *StrIdentifier*
→ *ArgumentList* **,** *StrIdentifier*
→ *ObjVarIdentifier*
→ *ArgumentList* **,** *ObjVarIdentifier*

NumericExp
→ **(** *NumericExp* **)**
→ *NumericExp*
→ *LogicalExp*
→ *LogicalUnaryExp*
→ *AdditiveExp*
→ *MultiplicativeExp*
→ *BitwiseExp*
→ *NumIdentifier*
→ *NumericLiteral*
→ *FunctionCommand*

LogicalExp
→ *NumericExp* **and** *NumericExp*
→ *NumericExp* **or** *NumericExp*
→ *NumericExp* **xor** *NumericExp*
→ *NumericExp* **==** *NumericExp*
→ *NumericExp* <> *NumericExp*
→ *NumericExp* **<** *NumericExp*
→ *NumericExp* **<=** *NumericExp*
→ *NumericExp* **>** *NumericExp*
→ *NumericExp* **>=** *NumericExp*
→ *StrIdentifier* **==** *StrIdentifier*
→ PointerVariable **== null**
→ **null ==** PointerVariable
→ **null ==** PointerTable **[** *NumericExp* **]**
→ PointerTable **[** *NumericExp* **] == null**

AdditiveExp
→ *NumericExp* **+** *NumericExp*
→ *NumericExp* **-** *NumericExp*

MultiplicativeExp
→ *NumericExp* ***** *NumericExp*
→ *NumericExp* **/** *NumericExp*
→ *NumericExp* **%** *NumericExp*

NotNumExp
→ *ObjVarIdentifier*
→ *StrIdentifier*

BitwiseExp
→ **bitnot** *NumericExp*
→ *NumericExp* **bitand** *NumericExp*
→ *NumericExp* **bitor** *NumericExp*
→ *NumericExp* **bitxor** *NumericExp*
→ *NumericExp* **<<** *NumericExp*
→ *NumericExp* **>>** *NumericExp*

AssignmentStatement
→ NumericVariable **=** *NumericExp* **;**
→ NumericTable **[** *NumericExp* **]** **=** *NumericExp* **;**
→ StringVariable **[** *NumericExp* **]** **=** *NumericExp* **;**

PtrAssignmentStatement
→ PointerVariable **=** *PointableIdentifier* **;**
→ PointerVariable **=** PointerTable **[** *NumericExp* **]** **;**
→ PointerTable **[** *NumericExp* **]** **=** *PointableIdentifier* **;**

PointableIdentifier
→ **null**
→ & StringVariable
→ & *NumVarIdentifier*
→ & *ObjVarIdentifier*

StrAssignmentStatement
→ StringVariable **=** *StrExp* **;**
→ StringTable **[** *NumericExp* **]** **=** *StrExp* **;**
→ StringVariable **+=** *StrIdentifier* **;**
→ StringVariable **+=** **Chr (** *NumericExp* **) ;**

StrExp
→ *StrAdditiveExp*
→ *StrIdentifier*
→ **Chr (** *NumericExp* **)**

StrAdditiveExp
→ *StrExp* **+** *StrExp*

StrIdentifier
→ StringVariable
→ StringLiteral
→ StringTable **[** *NumericExp* **]**

NumIdentifier
→ *NumVarIdentifier*
→ *NumTableIdentifier*
→ *StringCharIdentifier*

NumVarIdentifier
→ NumericVariable

ObjVarIdentifier
→ ChartVariable
→ DigIoUnitVariable
→ MixedIoUnitVariable
→ *TableIdentifier*
→ *CommunicationHandle*

NumTableIdentifier
→ NumericTable **[** *NumericExp* **]**

TableIdentifier
→ NumericTable
→ StringTable

StringCharIdentifier
→ StringVariable **[** *NumericExp* **]**

NumericLiteral
→ Integer32Literal
→ Integer64Literal
→ FloatLiteral

LogicalUnaryExp
→ **not** *NumericExp*

ConditionStatement
→ *IfStatement*
   StatementListOrEmpty
  EndifStatement
→ *IfStatement*
   StatementListOrEmpty
  ElseStatement
   StatementListOrEmpty
  EndifStatement
→ *IfStatement*
   StatementListOrEmpty
  ElseIfList
  EndifStatement
→ *IfStatement*
   StatementListOrEmpty
  ElseIfList
  ElseStatement
   StatementListOrEmpty
  EndifStatement

IfStatement
→ **if (** *NumericExp* **) then**

ElseStatement
→ **else**

ElseIfStatement
→ **elseif (** *NumericExp* **) then**
   StatementListOrEmpty

ElseIfList
→ *ElseIfStatement*
→ *ElseIfList ElseIfStatement*

```
EndifStatement
→ endif

CaseList
→ CaseStatement
→ CaseStatement DefaultStatement
→ CaseList CaseStatement
→ CaseList CaseStatement DefaultStatement

DefaultStatement
→ default :
      StatementListOrEmpty
    break

CaseStatement
→ case NumericExp :
      StatementListOrEmpty
    break

SwitchStatement
→ switch ( NumericExp )
      CaseList
    endswitch

ForStatement
→ for NumericVariable = NumericExp to NumericExp step NumericExp
      StatementListOrEmpty
    next

WhileStatement
→ while ( NumericExp )
      StatementListOrEmpty
    wend

RepeatStatement
→ repeat
      StatementListOrEmpty
    until NumericExp ;
```

Index