2

How Lookout*Direct* Works

This chapter explains the basics of how Lookout*Direct* works, including descriptions of objects, data members, connections, processes, and services.

Lookout*Direct* is a powerful yet easy-to-use Human-Machine Interface (HMI) and Supervisory Control and Data Acquisition (SCADA) software package for industrial automation. Lookout*Direct* runs under Windows and communicates with field I/O from Programmable Logic Controllers (PLCs), Remote Terminal Units (RTUs), and other devices. Typical Lookout*Direct* projects include continuous process monitoring and supervisory control, discrete manufacturing, batch applications, and remote telemetry systems.

Object-oriented and event-driven, Lookout*Direct* is a configurable package that requires no programming or scripting. Instead, you use Lookout*Direct* to create graphical representations on a computer screen of real-world devices such as switches, dial gauges, chart recorders, pushbuttons, knobs, sliders, meters, and then link your images to the actual field instruments using PLCs, RTUs, data acquisition boards, or other I/O devices.

Lookout*Direct* has many diverse capabilities such as Statistical Process Control (SPC), recipe management, Structured Query Language (SQL), built-in security, flexible data logging, running multiple processes on one computer, sophisticated animation, complex alarming, radio and dial-up telemetry support, audit trails of events and setpoint adjustments, multimedia support, touch screen compatibility, networking (including multiple client and server processes running on one or many computers), Dynamic Data Exchange (DDE & NetDDE), and more.

With Lookout*Direct* you can develop an application completely online, without shutting down. You do not have to recompile or download a database every time you make a modification, nor do you have to switch back and forth between programs. You do not even have to run separate development and configuration programs. Instead, you can add, delete and modify control panels, logic, graphics, PLCs, RTUs, I/O, and other field devices without ever interrupting your process.

Because Lookout*Direct* is object-oriented and event-driven, you can use LookoutDirect with other programs in the Microsoft Windows multitasking environment. For example, while Lookout*Direct* monitors and controls your

process, you can use a spreadsheet to analyze production figures of hourly average flow rates, then start a word processor to generate a memorandum, paste the spreadsheet into the memo and send it to a laser printer.

The remainder of this chapter describes Lookout*Direct* architecture—the components and how they work together. This will help you understand fully how to use Lookout*Direct* for all your continuous process, discreet, or batch applications.

Architecture

Once you understand the basic Lookout*Direct* components and the fundamentals regarding object-oriented and event-driven structure, using the program becomes much easier.

Lookout*Direct* consists mostly of objects and their data members, connections, and services. Developing a Lookout*Direct* application is a matter of creating, configuring, and connecting objects. Objects are software representations of everything from potentiometers and switches to PLCs and RTUs connected physically to the computers you have running Lookout*Direct*. You then make connections between the software objects. Lookout*Direct*, and the Lookout*Direct* services handle the connections between your computer and various PLCs or other controllers; between your computer and various sensors; between your computer and other computers; and between your computer and the Lookout*Direct* database, Citadel.

Add to this the idea of client and server processes and you are well on your way to understanding the basic structure of all Lookout*Direct* applications.

What is an Object?

A Lookout*Direct* object is a self-contained software unit designed to do something specific in your HMI/SCADA application. What each object does is generally referred to as its functionality. Each object has a set of data members you can log to a predefined database, and a set of parameters. The following diagram depicts the functionality, data members, and parameters of an object.

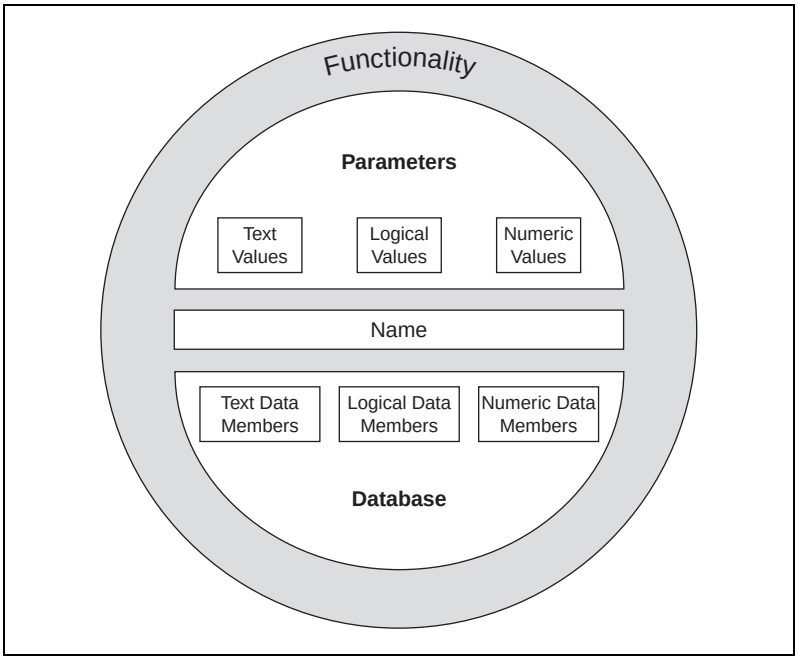


Figure 2-1. Objects Encapsulate Data Members, Parameters, and Functionality

Think of an object as a software model of something physical. For example, a potentiometer is something physical. You can adjust it up and down. In Lookout*Direct*, a Pot object represents the physical potentiometer. You can adjust it too.

Parameters define the limits of object functionality. For example, parameters set the minimum and maximum values of the Pot, the size of the smallest interval of adjustment, and other elements of the Pot's functionality.

The object *data members* contain information about the current state of the object, such as the value, whether the control is visible or hidden, and so on. The *database* can store data member information depending on what you want recorded, at what level of detail.

Lookout*Direct* Object Classes and Functionality

A Lookout*Direct* object is a specific instance of a Lookout*Direct* object class. You can think of an object class as the generalized form for an object. When you create an object, you are taking the general form described by an object class, defining it with specific parameters, giving it a name, and putting it to work as a software object. You can make as many objects as you need from each LookoutDirect object class, each specifically configured to perform the task you need that particular object to perform.

For example, Lookout*Direct* has both Pot and Switch object classes, from which you might create 20 pots and 30 switches. In this case, you would be creating a total of 50 objects using only two object classes.

Different object classes are designed to perform different functions, or tasks. For example, the Pot (potentiometer) object class operates differently from the Switch object class. This is the *functionality* built into every object class.

Global object classes are a special kind of object class. Each contains global system data such as the number of currently active alarms. You cannot create, modify or delete a global object, but you can use its data members just as you would use any other object data members.

When you create or open a Lookout*Direct* process file, Lookout*Direct* automatically creates three global objects: $Alarm, $Keyboard, and $System.

Functionality is the way an object class works, operates, or performs a task. Functionality is a general concept that applies in the same way to all objects in a given object class. Parameters, however, can be unique, and define the specific functionality of an individual object.

The object class definitions, found in the *LookoutDirect Object Reference Manual*, outline the functionality of each object class.

Parameters

An object's parameters define its characteristics. Lookout*Direct* uses object parameters to complete the definition of the object functionality. For example, **Data rate**, **Parity**, and **Stop bits** are a few of the parameters that define how a Modbus object works. Other examples include the **Control security level** of a Switch object; **Minimum**, **Maximum**, and **Resolution** of a Pot object; and **Data** of an Average object.

Every object class supports a set of parameters that you must fill in or select when creating a new object. Many parameters are *expressions*, which means you can change parameters programatically. Others require constant values. Some require you to pick specific settings.

Parameters that accept expressions appear as yellow data entry fields. These parameters can receive signals (that is, they are writable). See Chapter 1, *Expressions*, of the *LookoutDirect Developer's Manual* for more detailed information on expressions.

All the parameters for any given class are visible in the object definition dialog box. For information on how to create this object, see the *Creating & Managing Communication Links* section of Chapter 4.

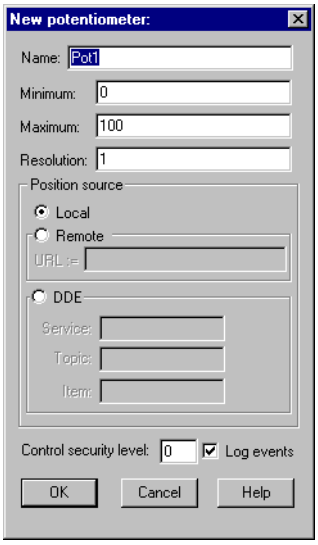


Figure 2-2. Pot Definition Dialog Box

Object Databases and Data Members

Each object has its own built-in database. The individual parts of this self-contained database are called *data members*. Some object classes have a very limited database, while others have extensive databases. The database of an object representing a PLC might have hundreds of data members; but a Switch object database has only six data members. You do not have to build a database—the data members are automatically available when you create the object.

In the case of a switch, the implicit value of the object is a part of the self-contained database. Data members can either generate (*write*) signals, receive (*read*) signals, or both.

Every data member contains a single value that can be one of three types: numeric, logical, or text. Lookout*Direct* is not strongly typed, however, so you can connect one type data member to another, and Lookout*Direct* will not generate an error message. See the *Data Polymorphism* section of this chapter for more information on how Lookout*Direct* interprets data of different types.

The built-in data members for each object are referred to as *native* members, and can be thought of as the default or automatic data members. You can add to and modify the database to suit your specific needs, attaching one or more aliases to a given data member, each with a different set of associated alarm, logging, or scaling properties.

Every object class has an explanation of its database in its class documentation, which you can access by clicking on the Help button in the dialog box used to create or modify an object. The following is an example of the switch object class database explanation.

Table 2-1. Switch Data Members

Data Members	Type	Read	Write	Description
(implicit)	logical	yes	no	Switch Position
enable	logical	no	yes	If TRUE (the default), enables DDE. If FALSE, disables DDE. The default value is ON. This input is ignored for non-DDE Switch objects.
reset	logical	no	yes	While this value equals TRUE, the control will be set to the value in `resetvalue`.
resetvalue	numeric	no	yes	Sets the value a control will take when the reset data member transitions from FALSE to TRUE.
value	numeric	yes	yes	The current value of the control. If you have remoted this control, then `value` is the current value of the position source.
visible	logical	no	yes	When FALSE, the switch object cannot be seen on the display panel. When TRUE, the switch can be seen and controlled.

Logical Data Members

Logical data members contain a value that represents a binary or on/off state. A light switch is a logical device—it is either on or off.

Logical data members are typically used to control equipment that can be turned on and off, to indicate that a piece of equipment is running, or indicate whether a limit switch is open or closed.

The Switch object generates a logical signal that is on when the switch is up and off when the switch is down. In the same way, the Pushbutton object generates a logical value that is on while the pushbutton is depressed. The Pulse object generates a logical signal that turns on and off at prescribed intervals, creating a logical pulse.

The logical signals that some objects generate can be displayed graphically on a control panel. See Chapter 2, *Graphics*, of the *LookoutDirect Developer's Manual*, for more information on graphical displays.

Lookout*Direct* recognizes the following logical constants as expressions:

- Logical constants that represent an on state: `yes`, `true`, `on`
- Logical constants that represent an off state: `no`, `false`, `off`

Lookout*Direct* is not case sensitive, so case variations such as On, ON, or oN will all be interpreted as ON.

See the *Data Polymorphism* section of this chapter for information on how Lookout*Direct* interprets numeric or text data when it is passed to a logical data member.

Numeric Data Members

A *numeric data member* is a floating point number representing analog values such as tank level, pressure, flow rate, voltage, and temperature. Numeric data members also represent time, either as a time period (span) or as an absolute time (that is, a particular time of day/week/month/year).

The Pot (potentiometer) object generates a numeric signal compatible with the numeric signals that monitor and control the analog input and output points on a PLC Numeric signals range from -1.7×10^{275} to 1.7×10^{275}.

See the *Data Polymorphism* section of this chapter for information on how Lookout*Direct* interprets logical or text data when it is passed to a numeric data member.

Numeric constants are entered using decimal digits (0 – 9), the minus sign (–), the exponent symbol (E or e), and the time format separator (:).

Some examples of numeric constants would be as follows:

- 0
- –123.779999
- 1.5E7 (15,000,000)
- –3.7E-3 (–0.0037)
- –.0036
- 23356636.234579

Time or *Time signals* are stored by Lookout*Direct* as numeric values that represent days and fractions of a day. For example, you enter one hour as 1:00:00. Lookout*Direct* interprets the number to the right of the rightmost colon (:) as seconds, the number to the right of the second colon from the right as minutes, the next number as hours, and the number to the left of the third colon from the right as days. If there are no colons in the entry, the time period is assumed to be given in days.

When your operating system switches in or out of daylight savings time, Lookout*Direct* corrects for the change relative to universal time so that there is no data discontinuity or loss in the Citadel database.

Table 2-2. Examples of Lookout*Direct* Time Constants

Time Period Constant	LookoutDirect Interpretation
0:23	23 seconds, or 0.0002662 days
75:00	75 minutes, 0 seconds, or 0.05208 days
12:00:05:01	12 days, 0 hours, 5 minutes, 1 second, or 12.003484 days
199::	199 hours, 0 minutes, 0 seconds, or 8.2917 days
0:10.023	10.023 seconds, or 0.0001160 days
12.75	12.75 days
17:64:22.5	invalid number: because hours are specified, minutes must be ≤59

You may enter one hour as 1:00:00, but Lookout*Direct* stores the number as 0.04167 (or 1/24 of a day). Days are represented by the integer portion of the number. The number zero represents Jan. 1, 1900. You may also find it helpful to know that one second = 0.000011574 and one minute = 0.000694444.

Tip If you display a numeric signal on a control panel, Lookout*Direct* provides a long list of numeric formats to choose from. You can set the format when you first create the display, or by right-clicking on the display and selecting **Display Properties**. The different possible formats are listed in the following tables.

Table 2-3. General Numeric Format Examples

Numeric Format	Value	Display
(General)	123.78	Displayed as 123.78
(General)	123.789	Displayed as 123.789
Note: General format displays the value in the most compact form possible.		

Table 2-4. Leading Zeroes Numeric Format Examples

Numeric Format	Value	Display
000000000	123.789	Displayed as 000000124
00000000	123.789	Displayed as 00000124
0000000	123.789	Displayed as 0000124
000000	123.789	Displayed as 000124
00000	123.789	Displayed as 00124
0000	123.789	Displayed as 0124
000	123.789	Displayed as 124
00	123.789	Displayed as 124
0	123.789	Displayed as 124

Table 2-5. Fractional Numbers with Trailing Zeroes Format Examples

Numeric Format	Value	Display
0.0	123.789	Displayed as 123.8
0.00	123.789	Displayed as 123.79
0.000	123.789	Displayed as 123.789
0.0000	123.789	Displayed as 123.7890
0.00000	123.789	Displayed as 123.78900
0.000000	123.789	Displayed as 123.789000
0.0000000	123.789	Displayed as 123.7890000
0.00000000	123.789	Displayed as 123.78900000

Table 2-6. Exponential/Scientific Notation Format Examples

Numeric Format	Value	Display
0E0	123.789	Displayed as 1E+2
0.0E+0	123.789	Displayed as 1.2E+2
0.00E+0	123.789	Displayed as 1.24E+2
0.000E+0	123.789	Displayed as 1.238E+2
0.0000E+0	123.789	Displayed as 1.2379E+2
0.00000E+0	123.789	Displayed as 1.23789E+2
0.000000E+0	123.789	Displayed as 1.237890E+2
0.0000000E+0	123.789	Displayed as 1.2378900E+2
0.00000000E+0	123.789	Displayed as 1.23789000E+2

Table 2-7. Hexadecimal Format Examples

Numeric Format	Value	Display
0x0	123.789	Displayed as 0x7
0x00	123.789	Displayed as 0x7
0x000	123.789	Displayed as 0x07
0x0000	123.789	Displayed as 0x007
0x00000	123.789	Displayed as 0x0007
0x000000	123.789	Displayed as 0x00007
0x0000000	123.789	Displayed as 0x000007
0x00000000	123.789	Displayed as 0x0000007

You can also use numeric signals to represent absolute times and periods of time. Because dates and times are represented by numeric values, you can add, subtract, and include dates and times in expressions, just as you would any other numeric signals.

A time *period* represents a span of time or a duration. Time periods are indicated in hours, minutes, seconds, and fractions of seconds. Numeric formats that represent time periods are characterized by capital letters (that is, H rather than h).

Table 2-8. Time Period Display Examples

Time Format	Value	Display
H	0.4789	Displayed as 11 (hours)
H.H	0.4789	Displayed as 11.5 (hours)
H.HH	0.4789	Displayed as 11.49 (hours)
M	0.4789	Displayed as 690 (minutes)
M.M	0.4789	Displayed as 689.6 (minutes)
M.MM	0.4789	Displayed as 689.62 (minutes)
S	0.4789	Displayed as 41377 (seconds)
S.S	0.4789	Displayed as 41377.0 (seconds)
S.SS	0.4789	Displayed as 41376.96 (seconds)

Table 2-8. Time Period Display Examples (Continued)

Time Format	Value	Display
HH:MM	0.4789	Displayed as 11:29 (11 hours, 29 minutes)
HH:MM:SS	0.4789	Displayed as 11:29:36 (11 hours, 29 min, 36 seconds)
HH:MM:SS.S	0.4789	Displayed as 11:29:36.9
HH:MM:SS.SS	0.4789	Displayed as 11:29:36.96
HH:MM:SS.SSS	0.4789	Displayed as 11:29:36.960
MM:SS	0.4789	Displayed as 689:36 (689 minutes, 36 seconds)
MM:SS.S	0.4789	Displayed as 689:36.9
MM:SS.SS	0.4789	Displayed as 689:36.96
MM:SS.SSS	0.4789	Displayed as 689:36.960

Absolute dates and times indicate a specific moment in time. Lookout*Direct* stores all absolute dates and times as numeric signals. It uses the 1900 date system in which the number 1 corresponds to midnight, January 1, 1900. The number 2 corresponds to midnight, January 2, 1900 and so on. For example, the number 34491.5 represents noon, June 6, 1994.

Numeric formats that represent absolute times are characterized by lower case letters (for example, hh:mm instead of HH:MM).

Table 2-9. Absolute Date and Time Display Examples

Time Format	Value	Display
hh:mm	34668.7889	Displayed as 18:56 (6:56 p.m.)
hh:mm:ss	34668.7889	Displayed as 18:56:02
mm/dd hh:mm	34668.7889	Displayed as 11/30 18:56
mm/dd hh:mm:ss	34668.7889	Displayed as 11/30 18:56:02
mm/dd/yy	34668.7889	Displayed as 11/30/94
mm/dd/yy hh:mm	34668.7889	Displayed as 11/30/94 18:56
mm/dd/yy hh:mm:ss	34668.7889	Displayed as 11/30/94 18:56:02
dd/mm hh:mm	34668.7889	Displayed as 30/11 18:56
dd/mm hh:mm:ss	34668.7889	Displayed as 30/11 18:56:02

Table 2-9. Absolute Date and Time Display Examples (Continued)

Time Format	Value	Display
dd/mm/yy	34668.7889	Displayed as 30/11/94
dd/mm/yy hh:mm	34668.7889	Displayed as 30/11/94 18:56
dd/mm/yy hh:mm:ss	34668.7889	Displayed as 30/11/94 18:56:02

Text Data Members

Text data members contain text character strings. These character strings consist of all displayable characters. You can use text signals to display alarm descriptions on the alarm panel, to display labels on a control panel, and in parameters or expressions. You can enter text signals as constants, or you can construct them with the many text functions available in expressions. Be sure to enclose text constants within double quotes (“ ”) when using them within expressions.

Some examples of text constants would be the following:

- “Water Temperature:”
- “ ” (empty text string)
- “Low level in ‘Polymer 2’ tank”
- “gpm”

See the *Data Polymorphism* section in this chapter for information on how Lookout*Direct* interprets logical or numeric data when it is passed to a text data member.

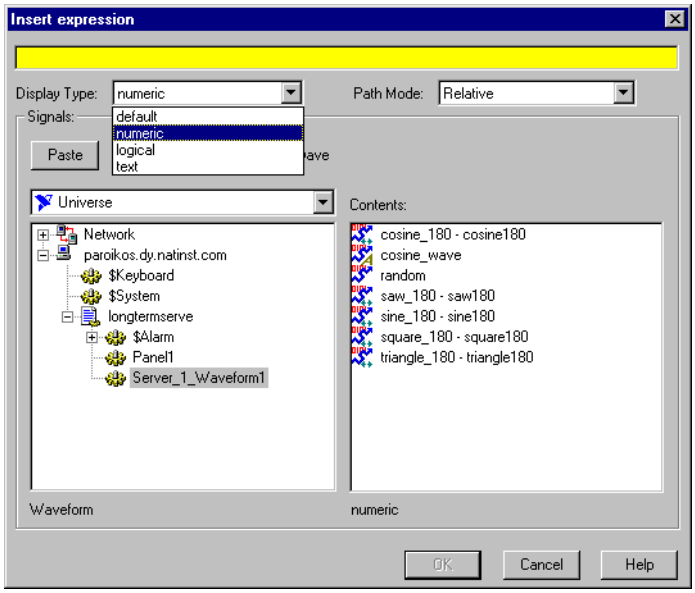
(implicit) Data Members

Many object classes have an (implicit) data member. This implicit value is either logical, numeric, or textual, depending on the object class, and follows the same rules that apply to all other data members. The implicit member represents what Lookout*Direct* considers to be the most commonly used data member of that object class. In many cases, it is the only data member of a class. It saves you time and reduces the amount of typing required to designate a data member.

For example, Lookout*Direct* could make you specify the numeric signal generated by a Pot object by typing `Pot1.numeric` where `Pot1` is the name and `numeric` is the current value of the Pot. Instead, you enter `Pot1` and Lookout*Direct* knows you are referring to the implicit value of the Pot. If you examine the Pot definition in the Lookout*Direct* online help, you will see that the (`implicit`) data member is the current value of the Pot.

Data Polymorphism

Polymorphic data is data that is not strongly typed. In Lookout*Direct*, data of one type is interpreted appropriately when connected to an input of another type. You can also select the data type when inserting expressions, using the selection box below the expression field, as shown in the following picture.



Logical values are displayed as 0 and 1 when interpreted as numeric values and as ON or OFF when interpreted as text values.

A numeric value of `0` displays as OFF when viewed as a logical value. Any value other than `0` is displayed as ON. A numeric value displayed as text shows the digits of the number.

Text strings displayed as numeric values appear as digits if the string consists only of digits in a valid Lookout*Direct* display format, such as a decimal number or a scientific expression. Numeric interpretation of text in a time format, such as `10:05:30` is interpreted as number (the fraction of one day represented by the time quantity) in scientific notation.

Any text string that does not consist of digits in a valid Lookout*Direct* format is interpreted as a 0, with the exception of `ON` or `TRUE`, which display as 1. `OFF` or `FALSE` are interpreted as 0, by default. The terms `On`, `Off`, `True`, and `False` are not case sensitive.

Text strings displayed as logical values are interpreted as a 0 or as OFF, except when the text string consists of a `1`, `On`, or `True`, all of which are interpreted as ON. Again, a text string consisting of `0`, `Off`, or `False` is interpreted as OFF, by default. Again, the terms `On`, `Off`, `True`, and `False` are not case sensitive.

Once you have placed an expression on your panel, you cannot change the data type in that expression. You have to delete that expression and place a new one if you want to change data types.

Expressions used as parameters in Lookout*Direct* objects do not permit you to select the data type because the object interprets input according to the data type required by that object. You can, however, use output of a data type different from that required by an object, as long as the output can be interpreted meaningfully by the object.

Polymorphic data types affect the DataTable object. In versions of Lookout*Direct* earlier than Lookout*Direct* 4.0, the DataTable used data members such as `A1.logical` to set the type of data in a particular cell. Some Lookout*Direct* developers used the DataTable to create, in effect, polymorphic data.

The Lookout*Direct* 3.*xx* DataTable data members are preserved in Lookout*Direct* 4 for the sake of compatibility. But it is not necessary to use the data-typed data members to simulate polymorphic data. To achieve this compatibility, however, Lookout*Direct* DataTable data is still strongly typed. You must input numeric data to DataTable numeric data members, logical data to DataTable logical data members, and so on.

Data Quality Attributes

Lookout*Direct* uses data quality attributes to keep track of any problems with your data. If there is a communications failure with a device, if the network connection to the source of the data is bad, if the data source is undefined, if the value is stale, or if some other problem arises, Lookout*Direct* creates an alarm to report the condition.

Because the alarm may only be visible on the local computer, and not on a computer accessing that data from another location, Lookout*Direct* places a red **X** over the affected control or expression when there is a data quality problem.

Check the Alarms panel of the process reporting the data quality problem for details on the problem. You can also use the data quality functions to monitor data quality and report problems. See the *Data Quality Functions* section of Chapter 1, *Expressions*, of the *LookoutDirect Developer's Manual*.

You can also set your computer to monitor alarms from processes running on other computers. See Chapter 9, *Alarms*, in the *LookoutDirect Developer's Manual* for information on how.

Connections

All the Lookout*Direct* objects you create either take the place of physical objects such as switches or potentiometers, or serve as your interface to some physical object connected to your computer, such as a PLC, and RTU, or another computer on the network that is connected to such objects.

You can connect all these objects in Lookout*Direct*, allowing signals to pass between them—much the same way you would wire a time delay relay to a motor starter relay, for instance. You can do this by connecting database members to each other, or by connecting database members to parameters.

In the broadest sense, one way to understand Lookout*Direct* is in terms of these connections. The following figure shows a data member to parameter connection, and a data member to data member connection.

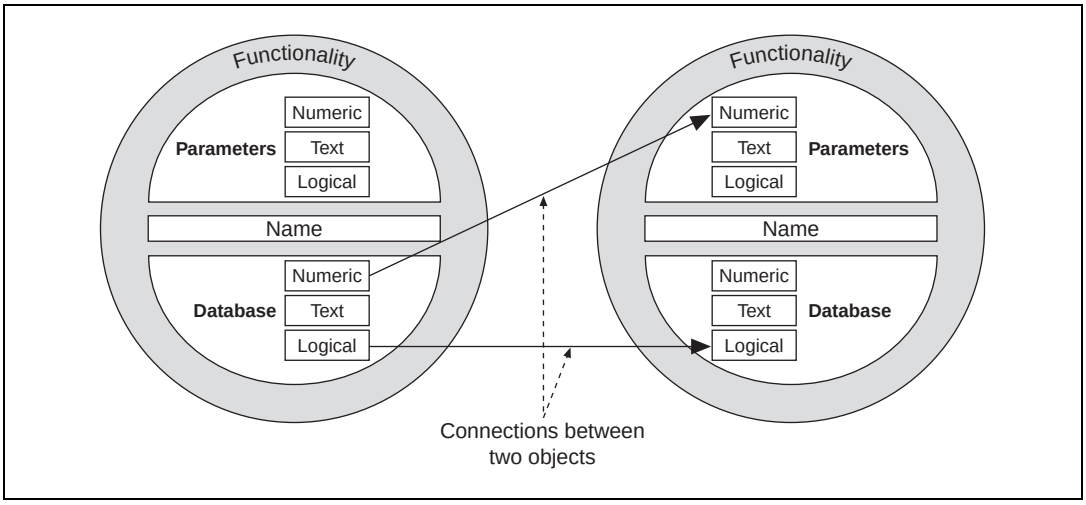


Figure 2-3. Example Connections Between Two Objects

For example, you might make the numeric data member of a Pot object the source for the **High Limit** parameter of an Alarm object. When you adjust the Pot, the Alarm **High Limit** changes.

In this way Lookout*Direct* handles the connections between your computer and various PLCs or other controllers; between your computer and various sensors; between your computer and other computers; and between your computer and the Lookout*Direct* database, Citadel.

Alarming, report generation, trending, data analysis, and a fast and flexible database are all built in; all you do is configure objects and data points, then create the connections, using the Lookout*Direct* dialog boxes.

Lookout*Direct* connections vary. Some control hardware directly by determining settings, some control or report on processes interactively, some analyze and present data. To all this capability, Lookout*Direct* adds a powerful yet simple way of making Lookout*Direct* connections across a network.

Connections and objects taken together are called a *process*. Because Lookout*Direct* runs multiple processes, and because networking has been made so simple, you can create as many or as few processes as you need, small or large, to increase the efficiency of your control systems.

Fortunately, you need not worry about creating drivers or databases or displays in Lookout*Direct*—all these things are either Lookout*Direct* objects, or contained in Lookout*Direct* objects. All you have to do is place objects where you want them, and make the necessary connections to and from them.

You do not have to worry about elaborate timing systems and polling loops, either. Because Lookout*Direct* is event driven, your connections report on and react to changes and other events as they happen, not in some arbitrary order built into your programs.

You can make the following kinds of connections in Lookout*Direct*:

- Direct—Made with the **Edit Connections** dialog box, these connections are fixed and work in one direction only. This sort of connection should only be made from a server process to hardware installed on the server's computer, or between objects contained within the same process.
- URL remote—A URL remote connection is a flexible type of fully reciprocal connection between a Lookout*Direct* control and a Lookout*Direct* object read/write data member. Changing the control changes the data member, as you would expect, but if the value of that data member changes from some other cause, the control value changes as well. This sort of connection is what you use to connect a client process to a server process, or to link two client processes through a server. (See the next section for an explanation of client and server processes.)
- Complex—You can make complex connections using variables and logic tests to respond dynamically to changing needs and circumstances. You can make direct complex connections, and with the Lookout*Direct* Symbolic Link object, make remote complex connections as well.

Client and Server Processes

Lookout*Direct* processes are programs you create to perform some specific function. Lookout*Direct* can run any number of processes at one time. You can open and close a process without disturbing other processes that are running at the time. Your processes can report and analyze data and control machinery, all while interacting with other Lookout*Direct* processes running on your computer and others scattered across the network.

It is important to make a distinction between client and server processes as you design and develop your Lookout*Direct* HMI.

- Computers running Lookout*Direct* servers should be properly connected to external industrial automation hardware such as FieldPoint from National Instruments. Computers running clients only should not be connected to external hardware. In other words, servers connect directly to hardware; clients connect remotely, through the server.
- Servers control and monitor; clients watch, report, and make adjustments to server settings. Servers can do all things clients do; clients act through servers.
- Client applications can move from computer to computer, and so must have no direct dependencies in order to run. In other words, nothing in a client application should refer directly to or depend on the computer running it. You should always use a remote source connection from a client control to a server rather than making a direct connection, so that the control reports settings as well as changing them.

As you might expect, you can have more than one client process attempting to alter a value in a single server process. When this happens, Lookout*Direct* accepts input as it arrives, so the first client input to arrive is executed before the next.

Supervisory Control

As you create and connect objects, you form a web, or system, containing many objects, all linked to perform a supervisory control strategy.

It is how you design your supervisory system, behind the control panel, that makes your process run. Your system routes signals from field components to bar graphs and visual indicators on control panels. It activates and deactivates alarms. You can design it to make complex decisions based on the values of field control signals and setpoints adjusted through pots and switches on control panels. You can include complex spreadsheet-style formulas as a part of your supervisory design.

Event-Driven Processing

An important concept to understand is that Lookout*Direct* is entirely *event-driven*, not *loop-driven*. To understand the significance of this design requires a digression, to explain how standard loop-driven programs work.

Loop-driven applications execute code sequentially from top to bottom, and loop back to the top to execute the same code over and over, as shown in Figure 2-4.

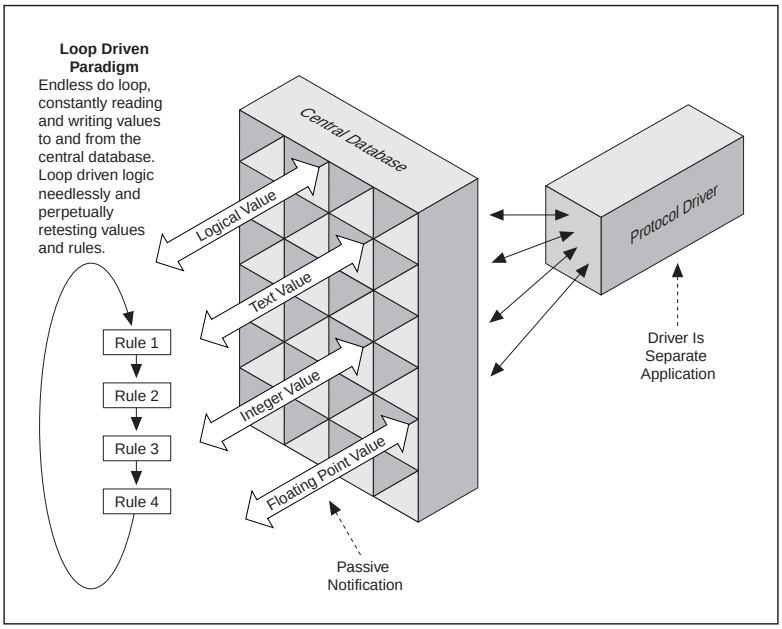


Figure 2-4. Example of Conventional, Loop-Driven Software Program

In this programming model, a given section of code or rule may execute millions of times before the result changes. This wastes computer processor time and slows down responses to frequent events.

The more rules you add to loop-based systems, the slower the response time. Also, as you add more names to the central database, speed and overall performance degrades. This is because many central databases use a *passive notification* system in which the rules of the loop-driven logic must scan an ever larger database for their appropriate values. The larger the database, the longer it takes the rule to find the data it needs to resolve its function.

In sharp contrast to this programming model, Lookout*Direct* is entirely *event-driven*. Each object remains quiescent, doing nothing until an event occurs, as shown in Figure 2-5.

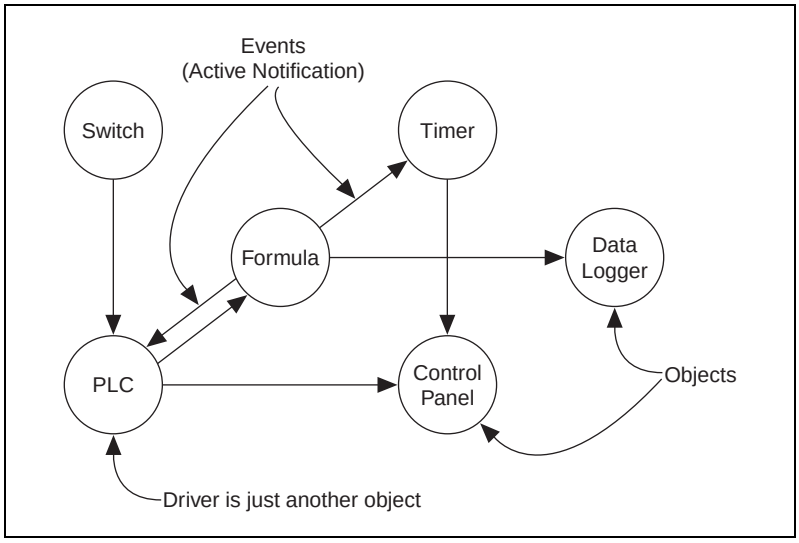


Figure 2-5. Lookout*Direct* Object-Oriented and Event-Driven Architecture

An *event* is a change in a data value. When an incoming signal changes, the object activates, processing the value according to its functionality. Objects only send out signals when the result of their processing changes. This is how an event propagates throughout your system, creating a chain-reaction that affects only the objects in the chain. This is called *active notification*.

Individual objects activate only when notified of an event. This event-driven paradigm closely simulates the responsiveness of physical switches, pushbuttons, and relays, and is the reason Lookout*Direct* is so fast.

Advantages of Active Notification

Consider the advantages of active notification over passive notification. Active notification is inherently event-driven while passive notification requires a constant do-loop to check for a change in a data value.

A good example would be two people trying to exchange information. One person could ask the other for information, and that person would respond. This would require one person to ask the other repeatedly if anything has changed. This is what loop-driven rules do when they constantly query the central database.

Alternatively, one person could just inform the other when something happens. This is what happens when you connect two objects in Lookout*Direct*.

Adding objects in Lookout*Direct* adds significantly less demand for processor time than adding new rules or enlarging the central database of a traditional loop-driven system.

Because Lookout*Direct* is entirely event-driven, the order in which you create the objects and connect them does not affect how your supervisory strategy works.

Lookout*Direct* Environment Services

While the cornerstone of object technology is the object itself, objects need an environment in which to function. Objects require the use of system resources like the serial port, hard disk, multimedia functions, and more. For example, multiple PLC objects may need to use the same communication port on your computer. In such a situation, Lookout*Direct* must provide an *environment service*—that is, a mechanism the objects can use to gain access to the communication port in an orderly and timely fashion.

An environment service, then, is a tool that Lookout*Direct* makes available to objects, or a function Lookout*Direct* performs outside of its object-oriented structure. Each environment service provides a special function. The following sections describe Lookout*Direct* environment services.

Serial Port Communication Environment Service

You can configure certain protocol object classes to represent and communicate with PLCs and RTUs through the serial ports of your computer. This environment service arbitrates serial port usage between objects representing PLCs and RTUs. For example, a single two-way radio connected to a serial port on the computer can communicate with several different brands of RTUs out in the field, each one using a different protocol. See Chapter 4, *Serial Communications,* in the *LookoutDirect Developer's Manual* for more information on configuring communications.

Database Environment Service

With the database environment service you can define or modify native data member parameters.

For example, the Modbus object class includes a native data member called 40001. You can give this native data member an alias such as `PumpSpeed`, and define associated unit scaling, alarming, deadband, and other parameters.

With Lookout*Direct* you can also directly import database information from external packages like Siemens APT. See Chapter 7, *Logging Data and Events*, and Chapter 8, *Structured Query Language*, of the *LookoutDirect Developer's Manual* for more information on database services provided by Lookout*Direct*.

Graphics Environment Service

Lookout*Direct* has an extensive library of standard graphics. These include various switches, potentiometers, pushbuttons, bar graphs, valves, tanks, pumps, and so on. You can also create your own custom graphic and add it to your Lookout*Direct* library. See Chapter 2, *Graphics*, of the *LookoutDirect Developer's Manual* for more information on graphics.

Alarm Environment Service

The alarm subsystem is a powerful and flexible mechanism for generating, displaying, logging, and printing alarms. This subsystem has several distinct parts including the alarm window, object parameters like **Alarm Group** and **Alarm Priority**, alarm filters, display parameters, and print settings.

Lookout*Direct* permanently archives alarms to disk. You can easily print this alarm history. See Chapter 9, *Alarms*, of the *LookoutDirect Developer's Manual* for additional information.

Multimedia Environment Service

Lookout*Direct* also provides a multimedia environment service you can use to play sound wave files. See the online help description of the Playwave object for more information on using this feature.

Security Environment Service

Lookout*Direct* has a highly sophisticated and comprehensive security system for both local and network security. You can configure control security, viewing security, and action verification. You can selectively determine which operators have control of what objects, which operators can view what control panels, and which objects prompt operators for verification of commands. See Chapter 6, *Security*, of the *LookoutDirect Developer's Manual* for more information on security.

Historical Logging Environment Service

With the logging environment service you can store real-time system information to disk in comma-delimited ASCII files, or in a special Lookout*Direct* database called Citadel.

The Lookout*Direct* Event Logger keeps track of who did what, and when they did it. Lookout*Direct* logs operator commands, from closing a process file to flipping a switch or adjusting a Pot. Along with each event, Lookout*Direct* logs the account name (operator), date and time of the event, name of the object adjusted, and the before and after settings of the object. See Chapter 7, *Logging Data and Events*, of the *LookoutDirect Developer's Manual* for additional information on logging environment services.

ODBC Environment Service

Because of the Lookout*Direct* Open Database Connectivity (ODBC) environment service, you can use other applications, such as Microsoft Access, to query the Lookout*Direct* historical database. See Chapter 8, *Structured Query Language*, of the *LookoutDirect Developer's Manual* for additional information on ODBC environment services.

DDE Environment Service

Lookout*Direct* can send its live process values to other applications, and it can receive real-time values from other applications. The Lookout*Direct* system acts as both a DDE client and a DDE server. See Chapter 5, *Dynamic Data Exchange*, of the *LookoutDirect Developer's Manual* for further information on DDE.

Networking Environment Service

Lookout*Direct* provides a full client-server networking environment service through the use of TCP/IP. With this environment service, you can monitor and control your processes from multiple workstations on a network. See Chapter 4, *Networking*, of the *LookoutDirect Developer's Manual* for information on networking Lookout*Direct* computers.

Redundancy Environment Service

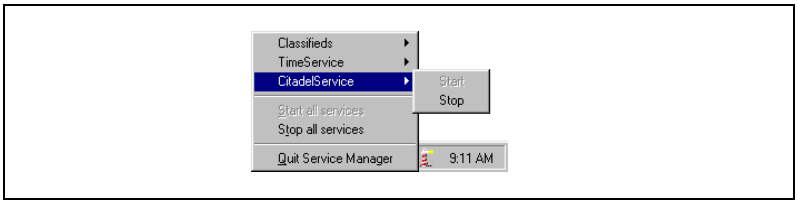
Use the Lookout*Direct* redundancy environment service to configure two computers for redundancy, providing automatic transfer of monitoring and control should one of the computers fail. See Chapter 10, *Redundancy*, of the *LookoutDirect Developer's Manual* for information on configuring computer redundancy.

Lookout*Direct* Windows Services

Lookout*Direct* requires three background services that run in Windows outside of the Lookout*Direct* application itself to be running on your computer while it is running: Citadel Server, Classified Ads, and Time Synchronization. In your Windows NT task manager, these services appear as *Classifieds*, *TimeService*, and *CitadelService.* Under Windows NT, these services run automatically as NT services. If you need to interact with these services, use the NT Services utility, found in **Start»Settings»Control Panel»Services**.

In Windows 98/95, Lookout*Direct* installs a *services manager* during installation, denoted by a small lighthouse icon at one end of your Windows task bar, as shown in the following illustration.

When you right-click on this icon you will see the following menu.



You can start or stop any of the Lookout*Direct* Windows services using this tool.