FactoryLink

Version 7.5

Task Configuration Reference Guide

Document Version: 2005-0425

Contents

Chapter 1 . . . . Introduction

The *Task Configuration Reference Guide* presents detailed technical information about how to configure FactoryLink tasks. The major audience of this guide includes users of FactoryLink who need to build a FactoryLink application.

It is recommended that you use this guide as a reference while you are developing your FactoryLink application.

TASKS BY FUNCTION

The tasks in this guide are arranged in alphabetical order. The following table provides a functional listing of the tasks.

Function	Task
Basic Functionality	Alarms
	Batch Recipe
	File Manager
	Math and Logic
	Persistence
	Print Spooler
	Programmable Counters
	Report Generator
	Run-Time Manager
	Scaling and Deadbanding

Function	Task
Database Tasks	Database Browser
	Database Logger
	Data Point Logger
	Database Schemas
	Historians • ODBC • Oracle • SQL Server • Sybase • dBASE IV
	PowerSQL
Graphics and Trending	Client Builder
	Trending
Networking	FLLAN
	PowerNet
	Virtual Real-Time Network and Redundancy (preferred for new applications)
Scripting	Math and Logic
Simulation	Waveform Generator and Sequencer
Timers and Counters	Event and Interval Timer
	Event Time Manager
	Programmable Counters

USING THIS GUIDE

General

Most of the tasks discussed in this guide use the Configuration Explorer. The information in this guide identifies the location of the tasks, defines the fields and parameters, and explains the usage of the tasks. For detailed information about the Configuration Explorer, see the *Configuration Explorer Help*.

Procedures that can be done using the Client Builder are mentioned, along with references to the *Client Builder Help* for detailed information.

The general organization of this guide is as follows:

- Overview
- Operating Principles
- Configuration Tables (how to access and field descriptions)
- Program Arguments
- Error Messages

Configuration Tables

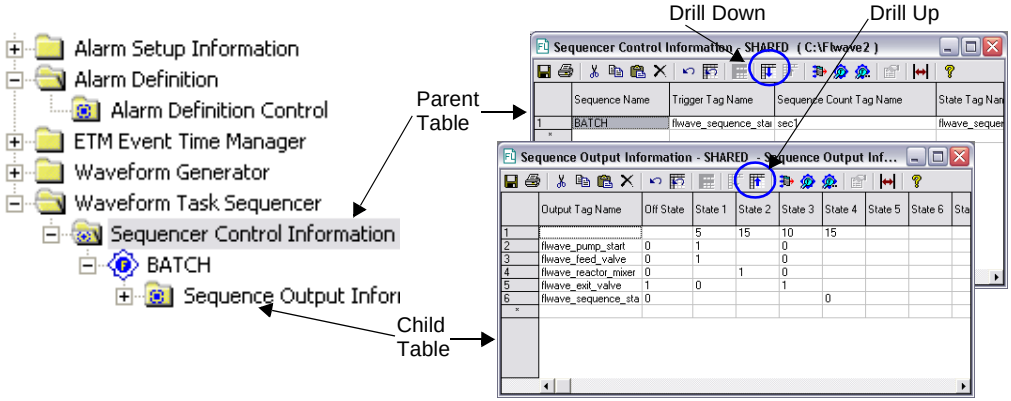
Accessing

In the Configuration Explorer, you can work with configuration tables in the Grid Editor or in the Form Editor. The principle method of showing the configuration tables in this guide is in the Grid Editor, but occasionally the Form Editor is used when it is easier to explain a function. Which editor to use is a user preference.

The Accessing section identifies the path to open the configuration tables. Many of the tables have a parent/child relationship. After a parent (control) table is set up, the child table becomes accessible. You can open the child table using either of these methods:

- Expanding the folders in the configuration tree and opening the appropriate table
- Using the Drill Down or Drill Up buttons in the toolbar.

Figure 1-1 Accessing Parent/Child Tables



For detailed information about using the Configuration Explorer and understanding the user interface, see the *Configuration Explorer Help*.

Field Descriptions

The Field Descriptions section provides a definition for each field that appears in the configuration table. With a configuration table open, you can obtain field-level help by clicking in a field and then clicking the Help button.

The field descriptions may show the valid entry, valid data types, and default values for the fields. The fields that do not show a default value usually are blank. An asterisk (*) before a field name denotes a field that accepts a tag name or constant value as a valid entry.

> **Tip:** When working in the configuration tables, you can specify the value for one field and click the Save button to have the default values automatically appear in the other fields. If a required field is not specified, a message identifying the required field appears.

Tag Names and Data Types

The configuration tables may require a valid entry of a tag name and a valid data type. The FactoryLink tasks use tag names to reference the tags in the real-time database. After a tag is defined, unlimited references can be made to it. A data type identifies the type of data that will be stored in the tag.

The *Fundamentals Guide* provides recommendations and guidelines for naming tags and a description about the data types.

Program Arguments

Program arguments are valid for the current revision of FactoryLink at the time of publishing. Not all listed arguments and their parameters may be implemented in earlier versions of FactoryLink.

Error Messages

The Error Messages section identifies the messages that may display on the Run-Time Manager screen if an error occurs with the task during run time. In some cases, error messages are also written to a log file. The location of this file is identified in the appropriate tasks.

In some error messages, references to tags and elements are synonymous and are used interchangeably.

TECHNICAL SUPPORT

If you have questions, contact your authorized Tecnomatix reseller or representative. For additional support information, go to **http://www.support.tecnomatix.com**.

- **1 | INTRODUCTION**
- *Technical Support*
-
-

Chapter 2 . . . . Alarms

The alarms task is used to define alarms and monitor them throughout an alarm cycle until the tag value no longer meets the alarm criteria.

Alarming interacts with the historian task to write alarm records to a database. The alarm data is logged to the relational database and/or to a file in a table or text format. The FactoryLink Distributed Alarm Logger performs logging as the status of the alarm changes: when the alarm occurs, when the alarm is acknowledged, or after an alarm has returned to the normal status.

At run time, the alarm task provides the operator the ability to view and manage the alarms which have met the established alarm criteria in the real-time database.

Alarming offers several useful features:

- Custom configuration of the alarm criteria
- Defining acknowledgment requirements
- Deadband provision in the alarm criteria
- Saving alarm records to a relational database or a flat file to preserve the time of alarm and alarm message for historical purposes
- Viewing alarms from multiple servers
- Customized Alarm Viewer to manage alarms as they occur, return to a normal status, and are acknowledged
- Enabling a run-time visual accounting of the alarm and the associated factory process using animations to represent factory subsystems
- Browsing alarm records previously logged to a database
- An operator logbook allows operators to add notes to an alarm to document why the alarm occurred, how it was corrected, or what is being done to correct it. These entries are saved in the relational database or a file when the alarms are logged.
- Notifying and acknowledging alarms using e-mail. Alarm e-mail notifications are sent to designated contacts, who are assigned the role of acknowledging alarms or receiving the notification for information purposes.

OPERATING PRINCIPLES

The Distributed Alarm Logger task monitors tag values in the real-time database and compares these values with the criteria defined by the developer in the configuration tables for the Distributed Alarm Logger task. You can establish the criteria that generate an alarm for any defined tag in the real-time database. If the value for the tag meets the criteria established for alarming, an alarm message is displayed on the Alarm Viewer for the operator. The operator monitors the alarm instances throughout the alarm cycle in the Alarm Viewer until the alarm tag values no longer meet the alarm criteria.

The alarm criteria can be configured to require an acknowledgment from the operator. The acknowledgment ensures the operator knows the alarm has been generated because the alarm does not clear from the viewer until it is acknowledged. If you want to preserve the times and occurrences of alarms, configure the Distributed Alarm Logger task to send the alarm data to a disk-based relational database using an historian task.

You can configure the Distributed Alarm Logger task to distribute the alarm messages across a network if you want the alarms to be viewed on more than one workstation. If the alarms are being logged and acknowledged, the node names where they were acknowledged are included in the alarm data sent to the relational database.

Alarm Logging Methodology

These steps describe how tags are monitored for alarm conditions and logged to a disk-based relational database.

1. The real-time database receives and stores tag values from various sources, such as a remote device, user input, or computation results from FactoryLink tasks.
2. The Distributed Alarm Logger task reads and compares the tag values stored in the real-time database with criteria defined in tables. These tables contain the configuration information for the Distributed Alarm Logger task.
3. When the value of the tag meets the criteria for an alarm, the Distributed Alarm Logger task sends the alarm to the alarm server for display on the Alarm Viewer.
4. Each time the tag value changes, the Distributed Alarm Logger task evaluates the tag. If the status has changed, the Alarm Viewer is updated.
5. When the value of the tag no longer meets the criteria for an alarm, the Distributed Alarm Logger task removes the alarm from the active alarm list. The alarm is cleared from the Alarm Viewer. However, if the alarm has been configured to require an acknowledgment from the operator, a status change to the alarm message occurs instead. The alarm is cleared from the list when it is acknowledged.

6 If the alarms are being logged to a relational database, the Distributed Alarm Logger task sends the alarm data to the relational database using a historian task each time a change occurs in the status of the alarm.

Establishing the Alarm Criteria

When the tag value meets the criteria to trigger an alarm, an alarm message is generated on the Alarm Viewer for the operator. An example of this is the alarming of the pressure in a fuel tank. A safe pressure is established as a reading of 0-1000:

- If the pressure exceeds 900, a HI alarm is generated to indicate the pending danger.
- If the pressure exceeds 950, which could be critical, the alarm generates a HIHI alarm.

In both cases, the tag condition is greater than (>) but each alarm is different. As the pressure changes, the display is updated to reflect the new readings and messages. When the pressure drops to 800, the danger passes and the alarms are no longer active.

The tag value must be checked against three components to establish this alarm:

- Limit—The limit is the value the condition is checked against. The example establishes the limit as 900.
- Condition—The condition that triggers the alarm. In the example, the condition is greater than.
- Deadband—The deadband is a range above or below the limit. The alarm stays active in this range. The example uses a deadband of 100 (900-100 = 800).

The Limit and the Deadband can both be set with a constant value or the value from another tag. The following valid condition settings generate alarms:

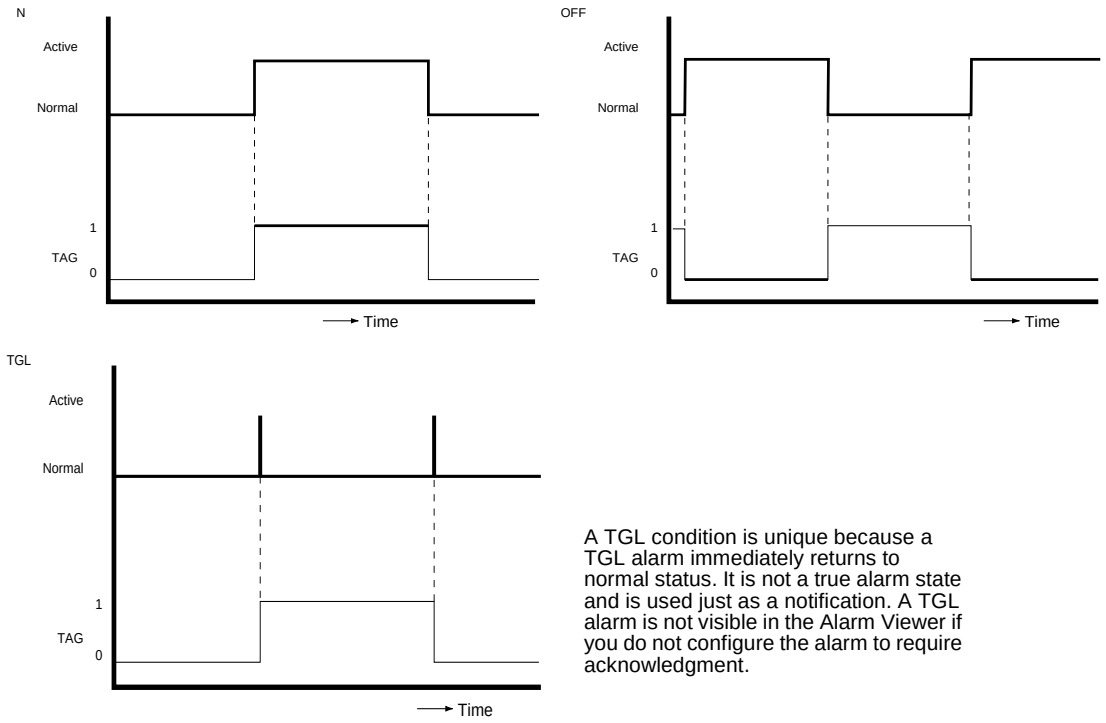
ON	An alarm is triggered when the value of the tag referenced is ON (1).
OFF	An alarm is triggered when the value of the tag referenced is OFF (0).
TGL	An alarm is triggered when the value of the tag changes, such as a change from ON (1) to OFF (0), from OFF (0) to ON (1), or if the change-status bits of the tag are set by a forced write.
HI, GT, HIHI or >	An alarm is triggered when the value of an analog or floating-point tag is greater than the value specified by the Limit.
LO, LT, LOLO or <	An alarm is triggered when the value of an analog or floating-point tag is less than the value specified by the Limit.
GE or >=	An alarm is triggered when the value of an analog or floating-point tag is greater than or equal to the value specified by the Limit.

LE or <=	An alarm is triggered when the value of an analog or floating-point tag is less than or equal to the value specified by the Limit.
EQ or =	An alarm is triggered when the value of an analog or floating-point tag is equal to the value specified by the Limit.
NE or <>	An alarm is triggered when the value of an analog or floating-point tag is not equal to the value specified by the Limit.

Digital Tags

The behavior of a digital alarm with specified limits by tag type is illustrated in Figure 2-2. The diagram represents an alarm status of active and normal based on a value, a limit, and deadband range.

Figure 2-1 Digital Alarm Cycle



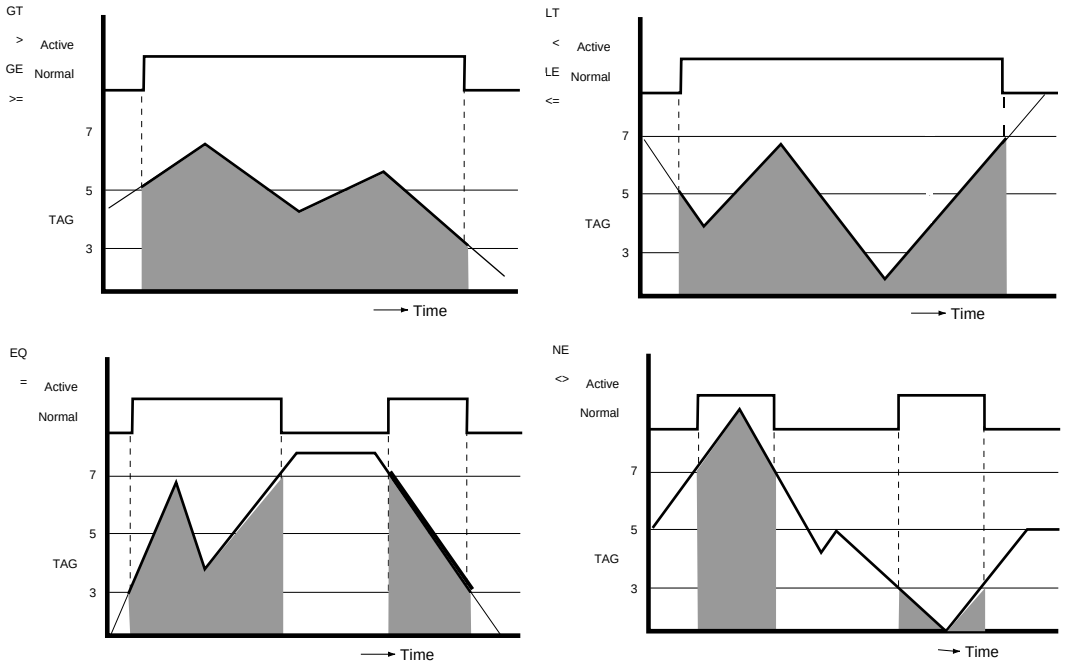
A TGL condition is unique because a TGL alarm immediately returns to normal status. It is not a true alarm state and is used just as a notification. A TGL alarm is not visible in the Alarm Viewer if you do not configure the alarm to require acknowledgment.

Analog, Long Analog, and Float Tags

The principles of operations are identical when operating on analog, longana or float tag types. The smallest unit detected is dependent on the type.

The behavior of an analog, longana, or float tag types with specified limits is illustrated in Figure 2-2. The diagrams represent an alarm status of active and normal based on the value, limit, and deadband range. All examples assume the Limit = 5 and Deadband = 2.

Figure 2-2 Analog and Float Alarm Cycles



Message Tags

When there is a change in the value of a Message tag the value is checked to be equal or not equal to the entire message defined as part of the alarm criteria.

Alarm Status

Every time the value of an alarm tag is changed, the new value is evaluated against the alarm criteria:

- If the tag value does not meet the alarm criteria, the status is considered normal.
- If the tag value meets the alarm criteria, a new alarm is added to the active alarm list and the status is active.
- If the alarm is already active and the value no longer meets the criteria, it returns to the normal status.
- If the tag returns to normal and the alarm does not require an acknowledgment, it is removed immediately from the list.
- If the tag returns to normal and the alarm required an acknowledgment and has been acknowledged, it is removed from the list.
- If the tag returns to normal and the alarm requires an acknowledgment and has not been acknowledged, it remains on the list until acknowledged and then is removed.
- The initialization of an alarm causes the Distributed Alarm Logger task to log the message to the relational database, providing the configuration is set to log alarms.

The Distributed Alarm Logger task maintains running counts of the number of alarms in the active queue at run time.

Alarm Categories

Categorizing alarms facilitates administration and analysis. Three methods are provided to show related alarms:

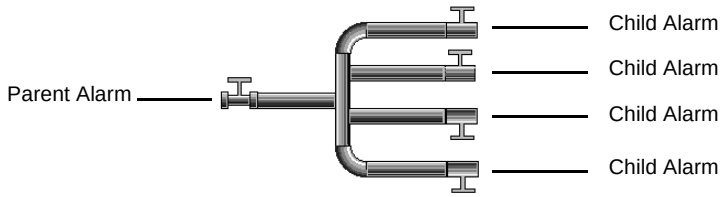
- Group Name—The group name is assigned to a class of alarms. Group names can be identifiers of the severity of the alarm, represent similar types such as pressure gauges, or indicate a combination of any other characteristics.
- Area—The area is assigned to each alarm individually. More than one alarm can reside in an area and alarms from different groups can also reside together. An area can reflect a physical location such as the boiler room or an area of responsibility such as maintenance.
- Priority—The priority is a numerical hierarchy assigned to each individual alarm. Use a number between 1 (lowest) and 9999 (highest) to set priority. Multiple alarms can be assigned the same priority number and multiple groups and areas can have common priority numbers within them.

At least one Group Name must be established to define any individual alarms. All alarms must belong to a group. The use of areas and priorities is optional. Categories enable filtering and sorting of alarms on the Alarm Viewer.

Parent/Child Relationship

The conditions which generate one alarm may also cause another related alarm to be generated. When these relationships exist, you generally do not want to display the additional alarms. For example, if the closing of a valve that feeds four different pipelines generates an alarm, it is a reasonable assumption that the lack of flow in each pipe would generate an alarm based on the value of the flowmeter tag as shown in Figure 2-3. These resulting alarms would not be important because you already know the flow has been cut off and why. This relationship between the alarms is identified as a parent/child relationship. In this example the main valve is the parent alarm of each of the flow alarms. The resulting child alarms are not displayed or counted as active alarms because they are a result of the parent alarm.

Figure 2-3 Parent/Child Alarm Relationship



However, if the main valve is open and one of the individual pipeline flowmeters registers an alarm, you would want to be advised. In this case the child is not dependent on the parent because the child alarm was initiated on its own. This alarm is displayed and counts as an active alarm.

Hide Alarms

Alarm hiding (sometimes referred to as masking) is done when you do not need to manage a particular set of alarms. Alarm hiding is used in the following common situations:

- Equipment maintenance
- Redundant systems
- Station functionality
- Bad sensor

Alarm hiding should not be confused with filters used with the Alarm Viewer. Alarm hiding can be configured to disregard a particular set of alarms for viewing and/or logging purposes. Alarm filtering selects specified alarms for viewing and suppresses other alarms from the Alarm Viewer; however, the alarms are still being logged and tracked.

Filtering is more common on multiuser or distributed systems. In these architectures, all users have the ability to monitor all alarms. However, certain operators may be responsible for a subset of these alarms. Filters enable operators to view only alarms they are responsible for on the Alarm Viewer.

If an alarm is hidden it does not act as a parent in parent/child relationships. To avoid potential problems when the parent alarm is hidden, child alarms also must be hidden.

Global Hide Tag

The Global Hide tag is used most frequently in redundant systems. In redundant systems, one node is the master and all alarms are active for this node (Global Hide tag = 0). The slave node or standby node has the Global Hide tag = 1.

Group Hide Tag

The Group Hide tag is used to hide equipment maintenance alarms. The developer must ensure that alarms are grouped by machine, so when a maintenance cycle begins, those alarms can be hidden.

There are three mask settings:

- Hide—hides alarm from the Alarm Viewer and the output alarm database
- Show—shows alarms on the Alarm Viewer and allows writing to the alarm database
- Event—hides alarm from the Alarm Viewer, but shows it in the alarm database

The Group Hide tags are also used to define station functionality. This is a special case because a node may have multiple functional requirements. For example, a node may function as a simple operator station for only one piece of equipment one day. The next day the same node may be the supervisor's station for all of the equipment. Groups are hidden based on the node functionality.

Individual Hide Tag

In some systems, individual alarms may need to be hidden to silence an alarm because of a malfunctioning sensor. When the sensor is repaired, the alarm needs to be monitored again.

Remote Group

There is no hiding function for alarms received from remote groups. Alarms should be hidden at the server node. If you do not want to view the alarms, create a filter in the Alarm Viewer so the alarms do not show.

Event Alarms

Event alarms are any alarms that are logged to a database but are not processed for viewing and acknowledgment. This provides the archival of the alarm condition without operator required processing. To configure an event alarm, use the Group Hide Tag or the Alarm Hide tag.

Locally Redefined Unique Alarm IDs

For networked systems, if a local Unique Alarm ID definition can be created and it matches a remote Unique Alarm ID, the alarm appears as if it is a local alarm. The Distributed Alarm Logger task must be running on the remote node. In addition to reporting alarms as if they were local alarms, this configuration provides the message text from the remote system to be displayed on the Alarm Viewer.

Alarm Persistence

Alarm persistence is the storing of current information about the status of active alarms and the child alarms at user-defined intervals. At startup the information is read preserving important information, for example, initial time and acknowledgment information.

Alarm persistence is activated by placing a **-w** on the program arguments field of Distributed Alarm Logger task line in the Shared domain and by configuring and running the Shared domain persistence task with a trigger in the Timed Save Trigger field, which causes the system to save the alarm persistence information each time the persistence trigger is set. The system then saves the Distributed Alarm Logger task information to the files:

- {FLAPP}/{FLNAME}/shared/al_log.prs
- {FLAPP}/{FLNAME}/shared/al_log.bak

If the ***.prs** file is unable to be read at startup, the ***.bak** file is used instead.

The al_log.prs file is updated at the time the Distributed Alarm Logger task is shut down and on a Persistence Timed Trigger change. The al_log.bak file is updated on a Persistence Backup Trigger change. For more information about the persistence function, see “Persistence” on page 377.

Upon restart of the Distributed Alarm Logger task, the al_log.prs or al_log.bak file is read into memory, and all alarms are checked for validity.

The active alarms are stored using their Unique Alarm ID number. If you have not defined a Unique Alarm ID in the alarm definition, one is defined at startup. If the configuration does not change, each alarm receives the same Unique Alarm ID as it did the previous time at startup. If the configuration changes, however, each Unique Alarm ID could be altered, and the Distributed Alarm Logger task could potentially load persistence information for incorrect alarms or not load persistence information.

Alarm Distribution

Alarms can be distributed over the network using the FactoryLink Local Area Network. Each node can share one or more groups of alarms with other nodes. The alarm is originated on the node it is defined on and is seen and acknowledged from other nodes that have been configured to receive information on that particular alarm. When the alarm is acknowledged, either at the source or at the remote, the source node accepts the acknowledgment and updates the new alarm status. All nodes receiving information on the alarm are updated.

Alarm Logging

If you want to preserve the time of alarm, alarm data, and the node that acknowledged the alarm, you can configure Distributed Alarm Logger task to read data from the tags in the real-time database and send the data to a disk-based relational database or to a text file. Data logged to a relational database is then available for browsing through the FactoryLink Database Browser or other browser program.

The Distributed Alarm Logger task logs data to a relational database using the same methodology as the FactoryLink Database Logger. The data is logged in a table format using a historian task. Alarm instances are logged at a status change: as the alarm occurs, when the alarm is acknowledged, or an alarm returns to the normal status. The tables for alarm logging output and their associated schemas are already defined for the Distributed Alarm Logger task.

If a remote group has logging turned on but no database information is defined on the client node, no information is logged. This condition does not result in the display of an error message.

When a remote node shuts down and restarts or reconnects after a communication failure with the same alarm still active, the logger tries to insert the alarm into the database twice. This condition results in generating a Duplicate Entry error.

The record length is determined by the size specified in the Message Size field of the Alarm Archive Control Information table in the Distributed Alarm Logger Setup table.

Table 2-1 and Table 2-2 describe the schema layout used to build the alarm entry table.

Table 2-1 Alarm Entry Table Schema

Column	Type	Array	Description
SEQ	NUMBER	11	Sequence Number of the alarm
AID	NUMBER	11	Alarm ID number
NAME	CHAR	48	Alarm TAG name
GRP	CHAR	16	Group alarm belongs to
AREA	CHAR	16	Area alarm belongs to
PRIO	NUMBER	6	Priority of the alarm
ITIME	CHAR	30	Initial Time (yearmodyhrmisc.mse)
ATIME	CHAR	30	Acknowledge Time (yearmodyhrmisc)
NTIME	CHAR	30	Normal Time (yearmodyhrmisc)
DUR	NUMBER	11	Alarm Duration in seconds
MSG	CHAR	162	Alarm Message
VAR1	CHAR	44	Variable 1 value at Initial Status
VAR2	CHAR	44	Variable 2 value at Initial Status
VAR3	CHAR	44	Variable 3 value at Initial Status
VAR4	CHAR	44	Variable 4 value at Initial Status
OPR	CHAR	8	Name of operator who acknowledged the alarm

Table 2-2 Alarm Entry Column Schema

Index	Unique	Column list
1	YES	SEQ + ITIME.

Logbook

Entries to the logbook are indicated by an asterisk in the Logbook field on the Alarm Control Viewer. The logbook data is viewable using the Database Browser. See *Client Builder Help* for more information on the Alarm Logbook.

CONFIGURING ALARMS

Alarms are configured in the server application and client project.This section explains how to locate the alarm tables, the definitions of the table fields, and the general design of the Alarm Viewer. Instructions for how to configure the Alarm Viewer are in the *Client Builder Help*.

The examples in this section are from the starter applications that are supplied with the software. These applications provide tables with preconfigured data to illustrate proper configuration of the fields. It is recommended that you use a starter application as the basis for your application. This will make configuration faster and easier.

Set Up Alarm Groups

The Alarm Group Control table is used to define the properties assigned to each alarm group. There must be at least one group defined for your application. All alarms must be a member of a group.

Color and sound information in the Alarm Group control table do not transfer to the Client Builder Alarm Viewer. These features are individually configured in the Client Builder application. If you are viewing alarms using ECS Graphics, colors and sounds can be configured in the Alarm Group Control table.

Three groups are preconfigured for default purposes: WARNING, CRITICAL, SYSTEM. These can be used or deleted as required.

Accessing

In your server application, open **Alarms > Distributed Alarm Definitions > Alarm Group Control.**

Field Descriptions

Note: The fields marked with an asterisk (*) are ***not*** passed to the Client Builder Alarm Viewer. They are only recognized by the alarm task in ECS Graphics.

Group Name — A string to identify the Alarm Group (required field).

Valid Entry: up to 16 uppercase alphanumeric characters

Group Text — Group message text that can appear with the output of each individual alarm message

An optional field but important to assist determining the alarm group when the message is output to a database or to the Alarm Viewer.

Valid Entry: up to 40 alphanumeric characters

*Group Composite Status Tag — Tag updated by the Alarm task that stores the code number representing the current status of all alarms in a particular alarm group

Valid Data Type: analog

Valid Entry: tag name
0: (IDLE) No alarms in the group are active.
1: (NORMAL) At least one alarm in the alarm group is unacknowledged and has returned to normal.
2: (ACK) At least one alarm in the alarm group is active and has been acknowledged.
3: (**ACTIVE**)At least one alarm in the alarm group is active and unacknowledged.
11: (NORM/ACK) At least one alarm in the alarm group is unacknowledged and has returned to normal and one other alarm is active and acknowledged.

*Group Number Active Tag — Tag updated by the Alarm task that stores the number of active alarms in this group

Valid Data Type: analog

Valid Entry: tag name

ACK — Indicates if the alarms belonging to this group need to be acknowledged

Note: If this field is set to YES or RST, the Unack Alarms Count Tag field in the General Alarm Setup Control table counts the unacknowledged alarms.

Valid Entry: **NO:** No acknowledgment required. The alarm disappears from the active list when it returns to normal.
YES: The alarm must be acknowledged.
RST: The alarm must be acknowledged but not until the alarm has returned to normal. This field can be used to reset alarms in the PLC or controller in conjunction with the alarm status.

Default: NO

*AUD — Determines if the alarms belonging to this group produce an audible signal when the alarm status is active

Valid Entry: **NO:** The alarms in this group will not produce an audible signal.
YES: The alarm produces an audible signal when it is active. The alarms in this group are included in the count maintained by the Audible Alarms Count tag in the General Alarm Setup Control table, which may be used to send a signal to an external device.

Default: NO

*Alarm Stat Print Dev

Print device number that corresponds to the line number of the printer device defined in the Print Spooler Information table.

If configured, alarm records are printed when generated or when there is a change in the alarm status.

Valid Entry: numeric print device number (Use 0 to disable)

Default: Blank, no printing is enabled.

*LOG

Specifies if group alarms are logged to a database or to a flat file

Valid Entry: **NO or N**: No logging.
YES or Y: When an alarm is changes status, it is logged to a relational database
FILE or F: When an alarm changes status, it is logged to a text or flat file.

Default: NO

*Log Method Tag

Tag that enables a run-time change of the logging method.

When set to the value 1, the alarm records are written to a database unless the database cannot be accessed. If this occurs, the alarm records are automatically written to a file as specified in the Device field of the Print Spooler Information table. To return to database logging, the operator must manually reset this tag to a value of 1. A run-time change is typically initiated using an animated graphic in the Client Builder program for an operator. A developer might use the Run-Time Monitor (RTMON) utility to reset the tag for troubleshooting purposes.

Valid Data Type: analog

Valid Entry: tag name
0: Run-time change of alarm logging is disabled.
1: Alarm logging is enabled to a database historian.
2: Alarm logging is enabled to a file.

Default: 0

*Initial FG Clr

Indicates the foreground color of an alarm in the initial status.

Valid Entry: a color or NONE

Default: Red

*Initial BG Color

Indicates the background color of an alarm in the initial status.

Valid Entry: a color or NONE

Default: Blk

*Initial Blink — Indicates whether or not the alarm blinks in the initial status. The speed may be chosen. YES blinks slowly.

Valid Entry: NULL, NO, YES, N, Y, SLW, FST
Default: No

*ACK FG Color — Indicates the foreground color of an alarm in the acknowledged status.

Valid Entry: a color or NONE
Default: Grn

*ACK BG Color — Indicates the background color of an alarm in the acknowledged status.

Valid Entry: a color or NONE
Default: Blk

*ACK Blink — Indicates whether or not the alarm blinks in the acknowledged status. The speed may be chosen.

Valid Entry: NULL, NO, YES, N, Y, SLW, FST
Default: NO

*Normal FG Color: — Indicates the foreground color of an alarm in the normal status.

Valid Entry: a color or NONE
Default: Yel

*Normal BG Color — Indicates the background color of an alarm in the normal status.

Valid Entry: a color or NONE
Default: Blk

*Normal Blink — Indicates whether or not the alarm blinks in the normal status. The speed may be chosen.

Valid Entry: NULL, NO, YES, N, Y, SLW, FST
Default: NO

*Group Hide Tag — Determines if the alarm messages are recorded and displayed for this group. At startup, some alarms can be generated that do not represent true alarm conditions. To avoid viewing these startup alarms, they are hidden. When startup is complete, the operator can select to process all alarms again. Event alarms are any alarms that are logged to a database but are not processed for viewing and acknowledgment.

Valid Data Type: digital, analog

Valid Entry: tag name
0: Alarms processed
1: Alarms not processed (hidden)
2: Event alarms (no filtering required)

Notification Group — Name for contact group that will receive an e-mail message about the alarms in this group.

Valid Entry: up to 80 alphanumeric characters (case-sensitive)

E-mail Subject — Short text message that appears at the beginning of the Subject line of an e-mail message transmitted by the corresponding Notification Group. This message is followed by the Alarm ID and Sequence Number.

Valid Entry: up to 48 alphanumeric characters (case-sensitive)

Define Alarms

Alarms are defined using two tables: The Alarm Definition Information table identifies the alarms associated with each group and the properties of each individual alarm, and the Alarm Relations Information table identifies the parent/child relationships between the alarms.

The basic definition of an alarm is to enter a tag name for the alarm identity and to establish the conditions which generate the alarm.

Note: Setup of the alarm group controls is essential before alarm records can be defined. All alarms must be defined within a group.

Accessing

In your server application open **Alarms > Distributed Alarm Definitions > Alarm Group Control > "group name" > Alarm Definition Information**.

Field Descriptions

Unique Alarm ID — A number that identifies the alarm record in the network

Each alarm on the network must be identified with a different number. If a number is not defined, a unique number is assigned by the al_log task. The assigned number is an internal number which does not appear in a tag field. It is important to note that the assigned number changes when the system configuration changes. If you define your own unique numbers, they will not change as the configuration changes.

This field is required for establishing Parent/Child relationships between alarms. If Parent/Child relationships are needed for **any** alarms, **all** alarm records in the application must have a Unique Alarm ID.

Valid Entry: 1 to 999999

Alarm Tag Name — Name of the tag to be evaluated for an alarm condition (required)

Valid Data Type: digital, analog, longana, float, message

Valid Entry: tag name

Cond. Determines the status of the alarm for digital alarms or TGL conditions. It also specifies the type of comparison of the alarm Limit value to the Deadband value in respect to current conditions.

Valid Entry: OFF: Off status or 0 for a digital tag
ON: On status or 1 for a digital tag
TGL:Changed status
LOLO < LO or LT: Less than the limit
HIHI > HI or GT: Greater than the limit
<= or LE: Less than or Equal to the limit
>= or GE: Greater than or Equal to the limit
= or EQ: Equal to the limit
<> or NE: Not Equal to the limit

Default: ON

If a TGL condition is established, the alarm vanishes as soon as it is detected because it immediately returns to normal. If the alarm is configured to require operator acknowledgment, the alarm is visible until it is acknowledged and then clears from the display. Alarms are logged to a relational database regardless of whether the alarm configuration requires acknowledgment.

Not all conditions are valid for all tag types. Table 2-3 shows the conditions supported by each tag type.

Table 2-3 Supported Conditions for Each Tag Type

Condition	Digital	Analog	Longana	Float	Message	Mailbox
ON	X					
OFF	X					
TGL	X	X	X	X	X	
LOLO LO LT <		X	X	X		
HIHI HI GT >		X	X	X		
LE < =		X	X	X		
GE > =		X	X	X		
EQ =	X	X	X	X	X	
NE < >		X	X	X	X	

Limit — A value used in conjunction with the Cond. and Deadband fields to determine an alarm condition. If a tag is defined for this field, it must be the same data type as the Alarm Tag Name and the Deadband field.

Valid Data Type: analog, longana, float, message, digital

Valid Entry: tag name or constant

Deadband — A value above and/or below the Limit value that determines an active alarm status.

The relationship of the Deadband value to the Limit is specified by the setting in the Cond. field. Once the alarm is triggered, it remains active until it moves past the deadband amount. If a tag is used, it must be the same data type as the tag in the Alarm Tag Name and the Limit field.

Valid Data Type: analog, longana, float

Valid Entry: tag name or constant

Message Text — Text that can be output to the Alarm Viewer, written to the alarm database, or output to a graphical animation.

Valid Entry: up to 160 alphanumeric characters

Variable (1-4): — Tags used as part of the alarm message when printed to a file, displayed in the Alarm Viewer, or written to a database. At run time, the values in these fields are substituted for the corresponding variable.

Valid Data Type: analog, longana, float, message, digital

Valid Entry: tag name

The Message Text field can include more than one variable, but the run-time display has constraints as shown in Table 2-4. Message text exceeding the maximum allowable number of characters per variable is truncated on the display.

Table 2-4 Variable Specifier Lengths in the Message Text Field

If the number of different VAn specifiers is...	The maximum number of characters displayed per specifier is ...
4	11
3	14
2	22
1	44

Embed the individual format variables ($VA1$, $VA2$, $VA3$, and/or $VA4$) in the text included in the Message Text field. For example, the Message Text field can contain the string: “The current status is $VA1$.” The tag in the Variable 1 field is Status tag. When the alarm is generated the current value of the tag in the Variable field, in this case the Variable 1 field, replaces the specifier when it appears on the Alarm Viewer. This value remains the same while the alarm instance is displayed on the Alarm Viewer.

To designate a variable that can be monitored for changing value, use format specifiers in the Message Text field instead of the VAn specifiers. Variable specifiers consists of two types:

- Ordinary characters, which are copied literally to the output stream
- Format specifiers, which indicate the format in which variable information will display

For information about format specifiers, see Appendix, “Format Specifiers.”

Priority Specifies the priority of the alarm for filtering and sorting purposes.

Valid Entry: 1 to 9999
Default: 1

Area Name Tag that provides an additional key for filtering and sorting purposes.

Valid Entry: up to 16 alphanumeric characters

Time-stamp Tag (Required if Time-stamp Format is specified) Message tag to specify the exact time an alarm status changed as a result of an operator process, such as acknowledgment.

If no tag is defined, time-stamping is determined when the change is detected by the Distributed Alarm Logger task.

Valid Data Type: Message

Valid Entry: tag name

Time-stamp Format Sets the time-stamp format of the alarm for the Time-stamp Tag field. If this field is specified, you must configure the Time-stamp Tag.

Valid Entry:
NULL: No formatting
USA: USA time format (YYMMDDHHMMSS)
EUROPE: European time format (DDMMYYHHMMSS)
USA_HS US: A time format with 1/1000 sec.
EUR_HS: European time format with 1/1000 sec.
USA_L: Long USA time format
EUROPE_L: Long European time format
USA_HSL: Long USA time format with 1/1000 sec.
EUR_HSL: Long European time format with 1/1000 sec.

Use Global Hide Used only when the Global Hide Tag in the General Alarm Setup Control table is set to ON. When an alarm is generated and the Global Hide Tag is set to 1 (ON), the alarm is not displayed. When an alarm is generated and the Global Hide Tag is set to 0 (OFF), the alarm is displayed.

Valid Entry:
Y or YES: Do not show alarm. (The Global Hide Tag field in the General Alarm Setup Control table must not be blank for this feature to work.)
N or NO: When an alarm is generated, the alarm displays.

Alarm Hide Tag — Tag indicates how an individual alarm is processed. Event alarms can be defined to log for tracking purposes but do not require viewing and acknowledging. Individual tags defined using this method require operator change to each record to allow processing.

Valid Data Type: digital, analog

Valid Entry: tag name, whose value at run time is:
2: Log to database; do not put alarm in active alarm list.
1: The alarm is not processed (not logged or put in active alarm list).
0: The alarm is processed (logged and put in active alarm list.

Status — Tag updated by the Alarm task that stores the current value of this alarm in the active alarm list. Interpretation of the status code values is dependent upon the tag type. See Table 2-5 for code definitions and values at run time.

Valid Data Type: digital, analog

Valid Entry: tag name

Table 2-5 Run-Time Status Values

Status	Definition	Analog Tag	Digital Tag
Initial	An active alarm; an alarm that has its alarm criteria met. The alarm remains in this status until there is operator action.	3	0
Acknowledged	The alarm status which occurs when the operator acknowledges an active alarm. The alarm remains listed on the Alarm Viewer after acknowledgment until the alarm criteria return to normal. (Only alarms in groups configured to require acknowledgment will remain listed on the Alarm Viewer after a return to normal status occurs.	2	1
Normal and not acknowledged	The criteria that triggered the alarm has been removed but the operator has not performed the acknowledgment function.	1	0
Idle	There is presently no criteria to trigger this alarm and the alarm does not need acknowledgment from prior alarm condition.	0	0

Notification Group — Name for contact group that will receive an e-mail message about the specific alarm.

Valid Entry: up to 80 alphanumeric characters (case-sensitive)

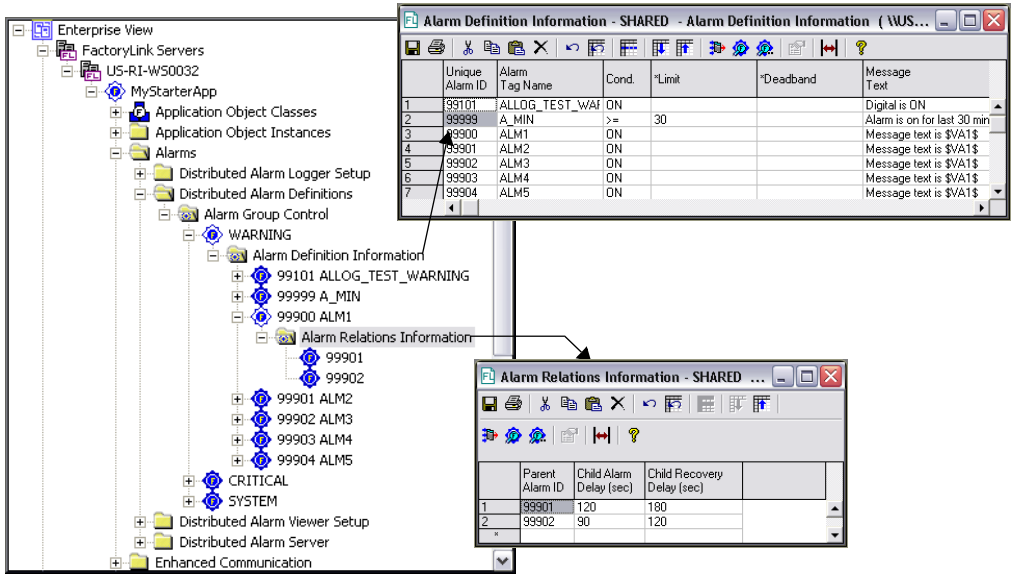
E-mail Subject — Short text message that appears at the beginning of the Subject line of an e-mail message transmitted by the corresponding Notification Group. This message is followed by the Alarm ID and Sequence Number.

Valid Entry: up to 48 alphanumeric characters (case-sensitive)

Define Parent-Child Relationships

The Alarm Relations Information table establishes a hierarchical relationship between alarms with related functionality as shown in Figure 2-4.

Figure 2-4 Parent-Child Relationship Tables



Parent-child alarm relationships are based on the parent alarm status. When a child alarm is initiated within the defined child alarm delay, it is hidden if the parent alarm is in the ACTIVE status. The child alarm is activated when the parent alarm returns to NORMAL. If the parent alarm is already in the NORMAL status, the child alarm is activated immediately.

Each alarm can have multiple parent/child relationships. Alarms defined in a remote group can never act as a child alarm. A parent alarm must have a defined Unique Alarm ID to create the child alarms on the local node.

Each alarm is evaluated by the Distributed Alarm Logger task and compared to its parent/child relationship prior to displaying.

- If the alarm is a parent, it is displayed.
- If the alarm is a child and the parent status is not active, the child is displayed.
- If the alarm is a child and the parent status is active, the child alarm is disregarded or displayed based on delay criteria you establish in the relationship.

In the parent/child relationship, two kinds of delays can be specified: child alarm delay and child recovery delay. These delays specify the time allowed between the generation or clearing of a parent alarm and the activation of a child alarm unrelated to the parent.

Child Alarm Delay

The length of time a child alarm is suppressed after a parent alarm is triggered is the child alarm delay.

The conditions that generate both the parent and child alarms must return to normal to allow the alarm statuses to return to normal. When both have returned to normal, the parent/child relationship is reestablished. At the next invocation of the parent, the timer is started again to inhibit the display of the child alarm for the child alarm delay period. These concepts are shown in Figure 2-5, Figure 2-6, and Figure 2-7.

Figure 2-5 Child Alarm Delay - Child is Suppressed

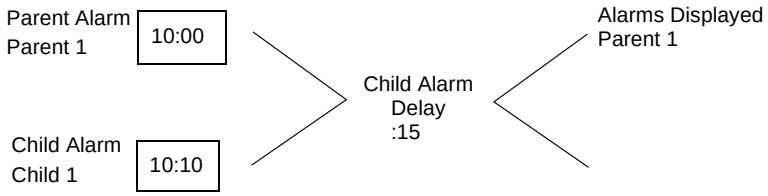


Figure 2-6 Child Alarm Delay - Child is Not Suppressed

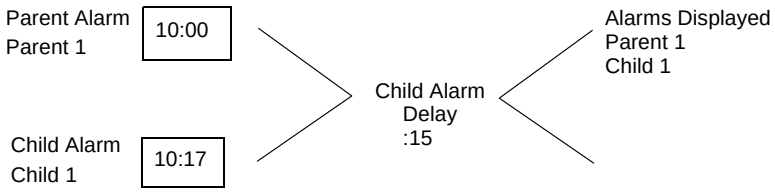
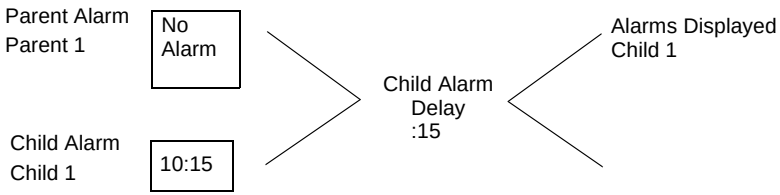


Figure 2-7 Child Alarm Delay - Child Alarm Only



Child Recovery Delay

The length of time a child alarm is provided to return to normal status after a parent alarm has been returned to normal status is the child recovery delay.

In the previous example, the main valve causing the generation of the parent alarm was shut off. This generated the four pipeline alarms but they are disregarded because they are redundant. If the main valve is now turned on, the flow should return to all four pipelines. The child recovery delay provides sufficient time for a child alarm status to return to normal. If the child status cannot return to normal in this time period then the child alarm generates an alarm.

After the child status has returned to normal, and the parent has a normal status, the parent/child relationship is reestablished. Figure 2-8 and Figure 2-9 illustrates these concepts.

Figure 2-8 Child Recovery Delay - Child Recovers

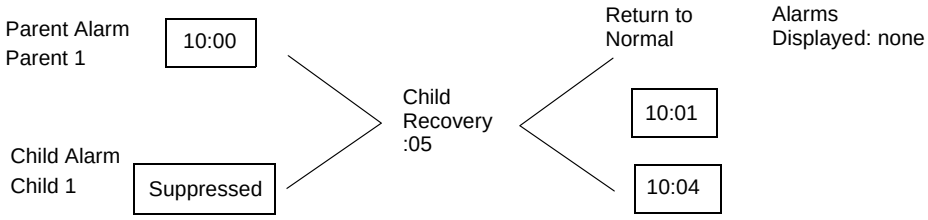
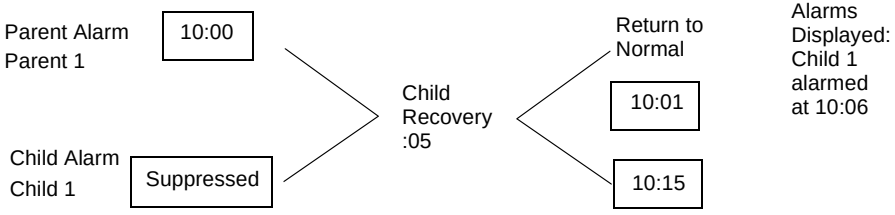


Figure 2-9 Child Recovery Delay - Child Alarms



TGL type alarms should not be configured as parent alarms. When a TGL alarm is generated it becomes ACTIVE and immediately returns to NORMAL. A TGL alarm never remains in the ACTIVE status. Using a TGL alarm as a parent would result in the child alarm never being hidden. An alarm can be a child to more than one parent alarm.

Parent Alarm ID Unique Alarm ID of the parent alarm. An entry is required for each additional parent if the child is subordinate to more than one alarm.

Valid Entry: The parent Unique Alarm ID, 1 to 999999

Child Alarm Delay Delay between the activation of the parent alarm and the activation of the child alarm. The parent always hides the child alarm if no delay time is entered. The child alarm is not displayed if a time is entered and the child alarm is activated within that period. The child alarm displays if a time is entered and the child is activated after that period.

Valid Entry: 1 to 30000 (seconds)

Child Recovery Delay Delay between the return to normal of the parent and child alarms. A child alarm is not displayed if the parent and child return to normal within the delay period. The child alarm is generated if the parent returns to normal and the child does not within that period.

Valid Data Type: Numeric

Valid Entry: 1 to 30000 (seconds)

Set Up Database Archive Requirements

Accessing

In your server application, open **Alarms > Distributed Alarm Logger Setup > Alarm Archive Control**.

The Alarm Archive Control table is used to identify the database or file where the alarm data is stored. Any relational database supported by FactoryLink can be used to log alarms. If you start a new application, default field names are provided to write to SQL Server or the internal dBASE IV file format. You must also configure the historian database tables to select the task which corresponds to the type of database you want to use to log alarm records.

Alarms are logged as soon as they are generated. The Logger task (AL_LOG) performs and controls all alarm logging. The internal database creates the files using the configured structure depending on the selection made at FactoryLink Installation.

To configure logging to a text file and archiving the text files, configure the Log File Trigger and Log File Directory files in this table and either the Log field or the Log Method Tag field in the Alarm Group Control table. Also, the Database Alias Name from the Alarm Archival

Control table must be entered in the Database Alias Name field on the appropriate Historian Information table if alarm data is logged to a relational database.

Field Descriptions

Database Alias Name Field contains an alias for the path and folder (directory) location of the relational database that stores the alarm data. The path for the alias is defined in the Historian Information table.

Valid Entry: up to 16 alphanumeric characters
Default: ALOG

Alarm Table Name Table name in the relational database that stores the alarm data. This name becomes a table name with the format of eight characters for the name and three characters for an extension. For example, the name ALARMS becomes a table name of ALARMS.dbf.

Valid Entry: up to 16 alphanumeric characters
Default: ALARMS

Logbook Table Name Table name in the relational database that stores the alarm logbook data.

Valid Entry: up to 16 alphanumeric characters
Default: LOGBOOK

Historian Mailbox Tag that stores each alarm message. If multiple messages are received, they are queued until they are written to the database.

This tag name must also be entered in the Historian Mailbox field in the Historian Mailbox Information table for the selected database.

Valid Data Type: Mailbox

Valid Entry: Mailbox tag name
Default: ALLOG_HIST_MBX

History Max Records Maximum number of records in a dBASE IV historian database. The oldest record is overwritten when the maximum number specified is reached.

Note: If a number is not specified, the records will continue to be written to the storage media until it is filled to capacity.

Valid Entry: 1 to 1000
Default: 1000

Message Size Size of the message column when it is saved into the alarm in the relational database.

If the message size is changed after the tables are generated, you need to alter or drop the existing tables from the database to prevent errors.

Valid Entry: 1 to 128
Default: 80

Timestamp Fields Data Type

Specify the data type to use when logging the alarms table timestamp fields.

Valid Entry: CHAR—Fixed length character string formatted as "yearmodyhrmisc.mse" where
year = Year
mo = Month
dy = Day
hr = Hour
mi = Minute
sc = Second
mse = Millisecond

DATE—Native date/time field, which may truncate the millisecond value depending upon the capacity of the database's native date/time field type.

Default: CHAR

Log File Trigger

Tag that initiates saving the currently queued alarms to a backup alarms text file. This tag is typically initiated using a graphical representation of the field which receives the trigger input. All the associated fields to save alarms to a file must be configured.

Valid Data Type: digital

Valid Entry: digital tag
0: OFF, do not save
1: ON, save current information to a permanent alarm text file

Log File Directory

Path used to save the backup alarm text file almmddyy.nnn.

Valid Entry: valid alphanumeric path name
Default: '{FLAPP}\{FLNAME}\{FLDOMAIN}

Set Up General Alarm Counters

The General Alarm Setup Control table is used to configure the tags that maintain counts of the alarms in the various states and the counts maintained by the Distributed Alarm Logger task. Default tag names are preconfigured for the user. The values of these tags can be monitored from a graphical representation of the system using Client Builder.

Accessing

In your server application, open **Alarms > Distributed Alarm Logger Setup > General Alarm Setup Control**.

Field Descriptions

Active Alarms
Maximum number of active alarms allowed. If more alarms are active than specified, an error message is displayed and the lowest priority alarm with the oldest time is removed from the list.

Valid Entry: 1 to 721
Default: 100

Global Hide Tag
Tag that hides alarms.

This tag works in conjunction with the Use Global Hide field in the Alarm Definitions table. Any alarm with the Use Global Hide field set to YES is hidden when this field is set to 1.

This field is used in conditions when reported alarms are not significant, for example, at startup time when alarms are reported because the application is not fully initialized. Alarms can be hidden until operations are stabilized.

Global hide cannot be activated if this field is blank.

Valid Data Type: digital

Valid Entry: tag name
0: Show and process alarms
1: Do not show or process alarms
2: Event alarms

Unack. Alarms Count Tag
Tag updated by the Alarm task that contains the number of alarms in the current unacknowledged status. This field is required for alarm acknowledgment.

If the ACK field of the Alarm Group Control table is set to YES or RST, a tag name in this field must be maintained.

Valid Data Type: analog

Valid Entry: tag name
Default: ALLOG_UNACK_COUNT

Active Alarms Count Tag

Tag updated by the Alarm task that contains the number of current active alarms.

Valid Data Type: analog

Valid Entry: tag name

Default: ALLOG_ACTIVE_COUNT

Audible Alarms Count Tag

Tag updated by the Alarm task that contains the number of unacknowledged alarms that have the audible flag set to YES.

If the AUD field of the Alarm Group Control table is set to YES, a tag name in this field must be maintained.

Valid Data Type: analog

Valid Entry: tag name

Default: ALLOG_AUDIBLE_COUNT

Print Active Alarms Tag

Tag that triggers the Distributed Alarm Logger task to update the file {FLAPP}/alarms.txt and print a list of all active alarms. (Alarms.txt is a default file name. Any name can be used.)

Valid Data Type: digital

Valid Entry: tag name
0: OFF
1: ON, print list

Default: ALLOG_PRINT_TRIGGER

Active List Print Device

Designates the print device or file where the active alarm output messages are routed. The number in this field corresponds to the line number of the print device identified in the Print Spooler Information table (For more information, see “Printing Alarms” on page 45.) Select the table line number that corresponds to the correct print device for the Alarm Stat Print Dev field in the Alarm Group Control table. An option to print to a file can also be selected using the same method. When printing to a file, the pathname of the file is defined in the device field instead of the device name.

Valid Entry: any positive integer
0: disables printing
Any other number is the line number of the print device.

Default: 0

Remote Notification Disable

(Optional) Name of a digital tag that enables or disables the remote e-mail notification feature, system-wide. If not specified or if specified and the tag is set to a 0, remote notification is enabled. If specified, and the tag is set to a 1, remote notification is disabled.

Valid Entry: tag name

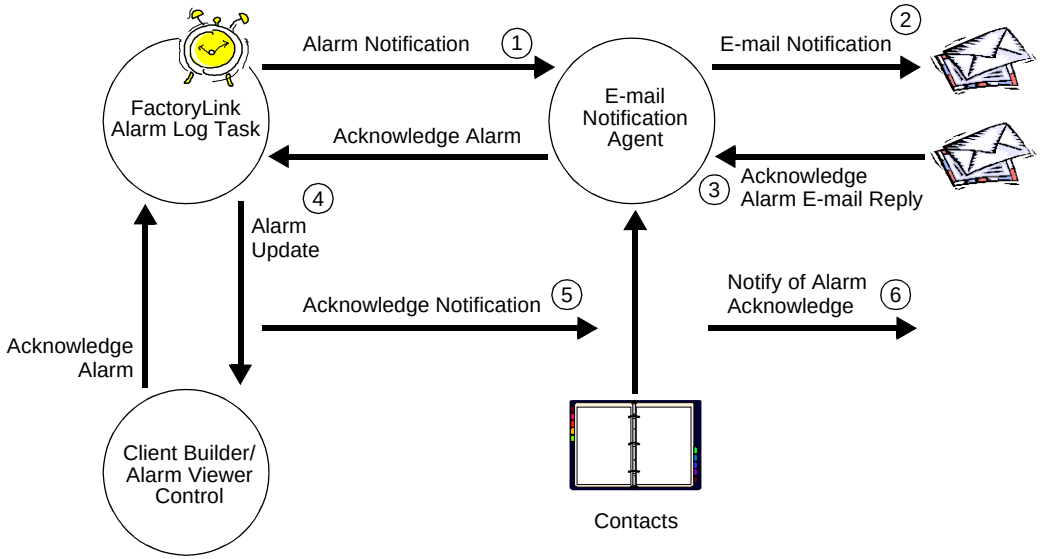
USING ALARM E-MAIL NOTIFICATIONS

An alarm or event defined in the alarm logger can be configured to use the e-mail notification agent to acknowledge an alarm by using an e-mail reply from contact recipients in a notification group. Whenever the alarm or event's status changes, an e-mail notification can be sent to specified contacts. Some contacts may be required to acknowledge an alarm while others need only be informed. If a contact fails to be notified within a specified amount of time, the alarm notification can be escalated to an alternate contact. Other contacts who are not required to acknowledge the alarm can be designated to receive an alarm notification.

If a client acknowledges an alarm in Client Builder, the acknowledge event is sent to the e-mail agent and the agent will notify all necessary contacts. Any pending outgoing e-mail pertaining to the original alarm will not get processed.

Figure 2-10 illustrates how an alarm notification is processed. ① shows a tag configured for alarms is in the active state. The alarm logger sends the alarm ID, sequence ID, and notification group information. At ②, the e-mail agent sends the alarm information to all contacts in the notification group. At ③, a recipient responds to the e-mail and acknowledges the alarm. At ④, the alarm logger verifies the contact is authorized to acknowledge the alarm and then formally notifies the alarm logger task that the alarm has been acknowledged. At ⑤, the alarm logger sends an acknowledge event to the e-mail agent with the alarm ID, sequence ID, and notification group. (The alarm logger task performs the actual acknowledgment.) At ⑥, the e-mail agent sends an acknowledgment e-mail to all contacts in the notification group.

Figure 2-10 Alarm Notification and Acknowledgment



E-mail Notification Messages

When the alarm logger notifies the e-mail agent of an alarm or event notification, it supplies the following information:

- Notification Group
- Alarm ID
- Alarm Sequence ID
- E-mail Subject Text
- Alarm Message Text

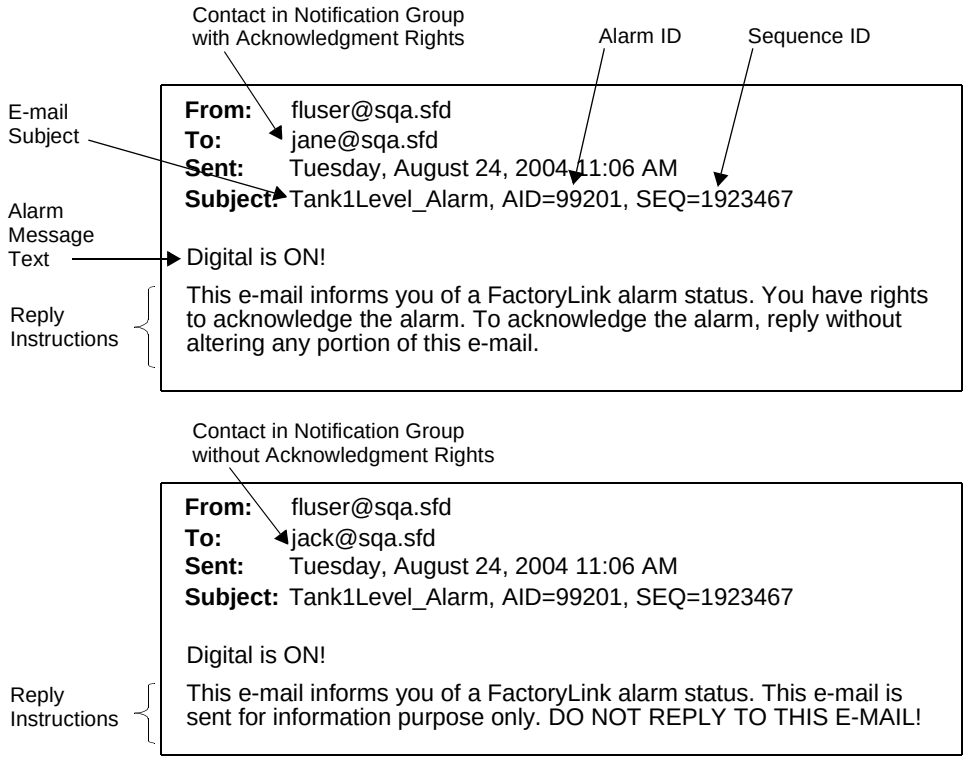
The notification group determines which contact groups will receive the e-mail message. The Alarm ID & Alarm Sequence ID make up part of the outgoing e-mail subject field. The outgoing E-mail Subject field contains the Subject Text + Alarm ID + Sequence ID.

The E-mail Subject text is defined in the alarm logger task. The Subject Text can contain any custom message. It is recommended that the Subject Text either contain the alarm tag name or descriptive text about the alarm; this information helps the recipient to identify the alarm.

An individual contact can be configured to receive the alarm message text as part of the e-mail body. Because some contacts may have restrictions for the size of e-mail they can receive (such as mobile phones), it is optional to include the alarm message text. The reply instructions are added to the message body of the outgoing e-mail only if the contact is configured to include the instructions. Figure 2-11 shows an outgoing e-mail message that requires an acknowledgment by the recipient and one that does not require acknowledgment (intended to simply inform the recipient).

The reply instructions are contained in a multilingual file named **emreply.txt**, located in FLBIN\MSG\[language] directory, where [language] is EN, FR, or DE. If a language is not supported, the desired language can be substituted in the text file for the currently defined language set using FLLANG. For example, if reply instruction must be sent to only Chinese recipients and the current FLLANG setting is EN (English), the EN text entry in the emreply.txt file can be changed to the Chinese text.

Figure 2-11 Alarm Notification E-mail Messages



E-mail Reply Messages

The e-mail agent checks its inbox every minute for responses sent by any contacts. When a contact recipient replies to an e-mail message, the e-mail agent verifies that the contact is configured to acknowledge alarms. If the contact is not configured to acknowledge alarms, the response is ignored and the active alarm is not acknowledged.

The e-mail agent uses the From and Subject field to match outgoing e-mail messages with the response messages. The Alarm ID and Sequence ID is used to determine what alarm will get acknowledged in the alarm logger task. Because this information appears in the Subject field, it is recommended that the contact not modify the Subject field when replying. Figure 2-12 shows examples of alarm acknowledged e-mail messages.

If a contact does not acknowledge an alarm within a specified time delay, the e-mail is escalated to another contact. More contacts are notified as time progresses. Escalation only

applies to alarms requiring an acknowledgment. If multiple contacts have the same delay time, the e-mail is sent to these contacts at the same time.

When e-mail clients are set to automatically reply to messages, the Subject field usually gets altered to include an automatic reply message. This type of response message is ignored because the response message fails to match the outgoing e-mail message Subject requirement.

Figure 2-12 Alarm Acknowledged E-mail Messages

Contact notified when alarm is acknowledged

From: fluser@sqa.sfd
To: jane@sqa.sfd
Sent: Tuesday, August 24, 2004 11:08 AM
Subject: Tank1Level_Alarm has been acknowledged, AID=99201, SEQ=1923467

Digital is ON!

This e-mail informs you of a FactoryLink alarm status. This e-mail is sent for information purpose only. DO NOT REPLY TO THIS E-MAIL!

Contact notified when alarm has changed but does not require acknowledgment

From: fluser@sqa.sfd
To: jane@sqa.sfd
Sent: Tuesday, August 24, 2004 11:16 AM
Subject: Tank1Level_Alarm, AID=99201, SEQ=1923467

Returned to normal!

This e-mail informs you of a FactoryLink alarm status. This e-mail is sent for information purpose only. DO NOT REPLY TO THIS E-MAIL!

Contact notified when event has changed but does not require acknowledgment

From: fluser@sqa.sfd
To: jane@sqa.sfd
Sent: Tuesday, August 24, 2004 11:45 AM
Subject: Egress Door Open, AID=1000, SEQ=1923500

Door is open!

This e-mail informs you of a FactoryLink alarm status. This e-mail is sent for information purpose only. DO NOT REPLY TO THIS E-MAIL!

E-mail Server Definition Table

This table defines the e-mail server parameters required to access the SMTP and POP3 mail servers. The e-mail agent will use these parameters to connect to the servers so that e-mail can be sent and retrieved.

Accessing

In your server application, open **Alarms > E-mail Notification Agent > E-mail Server Definition**.

Field Descriptions

Sender's Address Defines the e-mail address of the sender. The sender is usually an account set up to send e-mail from FactoryLink. Use of a user's personal account is not recommended because the sender's user name and password are visible in Configuration Explorer.

Valid Entry: up to 128 characters

SMTP Server Address Defines name or address of your SMTP server for outgoing e-mail.

Valid Entry: an address name (such as "mymailserver") or an IP address of up to 80 characters

SMTP Port The port number that supports the SMTP server. Obtain this information from your IT department or e-mail provider.

Default: 25

SMTP Logon Requires Secure Password Authentication? Indicates whether your SMTP server (outgoing mail) requires authentication to log in, which means the user name and password are to be encoded before being passed to the mail server. Most secure servers use authentication. Obtain this information from your IT department.

Valid Entry: NO, YES, N, Y

Default: NO

Note: If YES is selected and your system does not require authentication, your login will get rejected.

SMTP User Name Defines the user name account required by the SMTP server to log in.

Valid Entry: up to 255 characters

SMTP Password Defines the password required by the SMTP server to log in.

Valid Entry: up to 255 characters (case-sensitive)

POP3 Server Address
Defines the name or address of your POP3 server for incoming e-mail.

Valid Entry: an address name (such as “mymailserver”) or an IP address of up to 80 characters

POP3 Port
The port number that supports the POP3 server. Obtain this information from your IT department or e-mail provider.

Default: 110

POP3 User Name
Defines the user name account to log into the POP3 server. If this field is not specified, SMTP User Name field is used. (In most mail servers, the SMTP and POP3 login (user name and password) parameters are the same.)

Valid Entry: up to 255 characters

POP3 Password
Defines the password to log into the POP3 server. If this field is not specified, SMTP Password field is used. (In most mail servers, the SMTP and POP3 login (user name and password) parameters are the same.)

Valid Entry: up to 255 characters (case-sensitive)

Delete Mail From Server After Processing?
Indicates whether to delete the e-mail from the POP3 server (inbox) after an acknowledged alarm is successfully processed. A processed e-mail is considered an e-mail that has been validated as an acknowledgment to an active alarm. Deleting the processed e-mail messages frees up storage space and reduces the possibility that an old e-mail may get mistaken for a response to a current alarm.

Valid Entry: NO, YES, N, Y
Default: YES

Notification Groups Table

This table links the alarm logger to the e-mail agent. Each alarm group or alarm definition may reference a notification group, which is referenced in the Alarm Group Control table. When an individual alarm belonging to an alarm group changes, member(s) of the referenced notification group receive an e-mail message.

If a notification group is used on an alarm group level, e-mail is generated for *all* alarms in the group. If a notification group is defined on an individual alarm tag level, e-mail is generated only for that alarm or event tag.

Accessing

In your server application, open **Alarms > E-mail Notification Agent > Notification Groups**.

Field Description

Notification Group — The name assigned to a group of alarm tags or an individual alarm tag that determines which contact groups will receive the e-mail message. This group name must match the notification group name used in the Distributed Alarms Definitions tables.

Valid Entry: up to 80 alphanumeric characters (case-sensitive)

Contact Groups Table

This table defines the contact group names associated with the notification group. Each contact group is assigned a schedule (day and time) when the contacts in the group can receive e-mail notifications.

Accessing

In your server application, open **Alarms > E-mail Notification Agent > Notification Groups > "your notification group name" > Contact Groups**.

Field Descriptions

Contact Group — Defines the contact group name. that contains the e-mail addresses.

Valid Entry: up to 80 alphanumeric characters

Schedule Start (24hr Format) — Defines the start time that the contacts in a group can receive e-mail notifications. This time can coincide with the work hours of the contacts that belong to the contact group. The start time is expressed in 24-hour format.

Valid Entry: 0000 (midnight) to 2359 (11:59 p.m.)
Default: 0000

Note: It is possible to use a start time that is greater than the end time. For example, the group's availability to receive e-mail spans the time between two days such as an 8-hour shift that starts at 2200 (10:00 p.m.) and ends at 0600 (6:00 a.m.).

Schedule End (24hr Format) — Defines the end time that the contacts in a group are to stop receiving e-mail notifications. This time can coincide with the work hours of the contacts that belong to the contact group. The end time is expressed in 24-hour format.

Valid Entry: 0000 (midnight) to 2359 (11:59 p.m.)
Default: 2359

Note: It is possible to use an end time that is less than the start time. For example, the group's availability to receive e-mail spans the time between two days such as an 8-hour shift that starts at 2200 (10:00 p.m.) and ends at 0600 (6:00 a.m.).

SUN, MON, TUE, WED, THU, FRI, SAT — Indicates the day of the week that the contacts in a group can receive e-mail. The contacts can receive e-mail only on those days marked with YES or Y.

Valid Entry: NO, YES, N, Y
Default: NO – Sunday and Saturday
YES – All other days

Contact Definition Table

This table defines the contacts that belong to a contact group.

Accessing

In your server application, open **Alarms > Notification Agent > Notification Groups > "your notification group name" > Contact Groups > "your contact group name" > Contact Definition Information**.

Field Descriptions

Display Name Defines the display name associated with an e-mail address. The display name is usually the contact's first name and last name or a nickname.

Valid Entry: up to 80 alphanumeric characters

Address Defines the e-mail address for the contact

Valid Entry: up to 128 characters

Delay Before Notification (mins) Defines the time in minutes to wait until an e-mail is sent to a contact. This delay time is also known as the escalation time. All entries with the same time delay are sent an e-mail message at the same time.

Valid Entry: 0 to 9999
Default: 0

Log Escalation Indicates whether the e-mail notification agent is to log the escalation event.

Valid Entry: NO, YES, N, Y
Default: NO

Role Defines the role a contact plays when processing an e-mail response. The contact's role can be required to perform an acknowledgment or to be informed (no acknowledgment or response anticipated). If the contact responds to an e-mail that indicates an active alarm, the role for the contact is checked. If the role is set to acknowledge, the alarm logger is notified that the alarm was acknowledged.

A contact may have dual roles where the contact is informed of all alarm notifications and has the capability to acknowledge an alarm. If this case is desired, the contact's name must appear twice: one with the ACK role and the other with the INFORM role.

Valid Entry: ACK – Acknowledge
INFORM – Information (default)

Include Alarm Message? Indicates whether to include the alarm message in the e-mail message body.

Valid Entry: NO, YES, N, Y
Default: NO

Note: If the contact's e-mail provider has a message size limitation, the alarm message may exceed the limit. In this case, NO is recommended.

Include Reply Instructions? Indicates whether to include the special instructions about how to reply to an alarm in the e-mail message body.

Valid Entry: NO, YES, N, Y
Default: NO

Note: If the contact's e-mail provider has a message size limitation, the reply instructions may exceed the limit. In this case, NO is recommended.

Debugging the E-mail Agent

The alarm logger makes use of its "-ln" and "-dn" program arguments. Using "-ln" signifies logging to a file, and using "-dn" signifies displaying debug output to the console and DebugView.

The e-mail agent shares the use of these parameters for debugging. The "n" signifies a level:

1 – Informational (general messages)

2 – Configuration (configuration tables)

3 – Transaction messages between the e-mail agent and the mail server

4 – All (1-3)

When "–ln" is specified, the log file, Agent.txt, is created in:

%FLAPP%\flapp1\shared\shareusr\log

DebugView can also capture output to a file. DebugView is a freeware product from SysInternals (www.sysinternals.com).

PRINTING ALARMS

Printing to files, directing output to files and to the printer can be accomplished using several different methods. For an explanation of these methods, see Table 2-6. The formats used for printing can be modified.

Open the file **%FLINK%msg\{language}\al_fmt.txt** with your favorite text editor program. Edit this text as indicated in the file to configure the print formats and tokens. The guidelines for configuration of each option are included in each section of the file.

Table 2-6 Printing and Directing Output to Files Using the al_fmt.txt File

No.	Description	Configuration
1	Print to a file using the print spooler	In Alarm Group Control table, set the Alarm Stat Print Dev field equal to a printer line number in the Device field of the Printer Spooler table that contains the address of a file; example, C:\msg\filename.txt
2	Print to a printer	In the Alarm Group Control table, set the Alarm Stat Print Dev field equal to a printer line number in the Device field of the Printer Spooler table that contains the device port; example, COM1: or LPT2
5	Print all active alarms to a file	Print Active Alarms Tag field is set to ON. The Active List Print device tag field is set to a printer line number in the Device field of the Printer Spooler table that must contain the address of a file; example, C:\msg\filename.txt
6	Print all active alarms to a printer	Print Active Alarms Tag field is set to ON. The Active List Print device tag field is set to a printer line number in the Device field of the Printer Spooler table that must contain the device port; example, COM1: or LPT2:

ALARM VIEWER DESIGN

The Alarm Viewer is an alarm management tool that provides a visual display of run-time alarms in active states: Initial, Acknowledged, and Normal but not yet acknowledged. The Alarm Viewer is a customized ActiveX control that interfaces with the FactoryLink alarm server. The type, sorting, fonts, colors, blinking background or text, fields, field sequence, and filtering of the displayed alarms are designed according to user preferences. Each client workstation can maintain a separate design scheme for viewing alarms. Or, once an Alarm Viewer is configured, the design can be shared among other workstations.

Client Builder provides an integrated design and run-time environment. Accessed in Client Builder, the Alarm Viewer configuration can be modified in design mode and the changes observed immediately in the run-time mode, so the designer can make adjustments as needed to complete the design. Certain features or options can be locked to prevent operator changes at run time.

In addition to the Alarm Viewer, an Alarm Banner Viewer is configured from the same ActiveX control. The Alarm Banner Viewer, which displays up to three alarms, provides a subset of the Alarm Viewer features. Because of its smaller size, it is easily positioned on various Client Builder mimics. Depending on the design, this viewer shows the operator the most critical or newest alarms. See the *Client Builder Help* for instructions to configure the alarm viewers.

RUN-TIME ALARMING

As alarms are generated, the information is displayed on the run time Alarm Viewer or Alarm Banner Viewer. These alarms remain on the display until the alarm criteria no longer meets the defined alarm conditions. If an alarm is defined as an alarm that must be acknowledged, the alarm output remains listed even after the alarm condition is removed and the operator has not manually acknowledged the alarm.

The size of the viewers is determined at design time, but fields can be resized at run time. The horizontal scroll bar (when enabled in design mode) allows operators to view the columns (during run time) when they are no longer visible due to resizing. Other run-time features are sort, filter, acknowledgment of alarms, and printing. Figure 2-13 shows the Alarm Viewer with all of the basic features selected, and Figure 2-14 shows the Alarm Banner Viewer.

Figure 2-13 Parts of the Alarm Viewer at Run Time

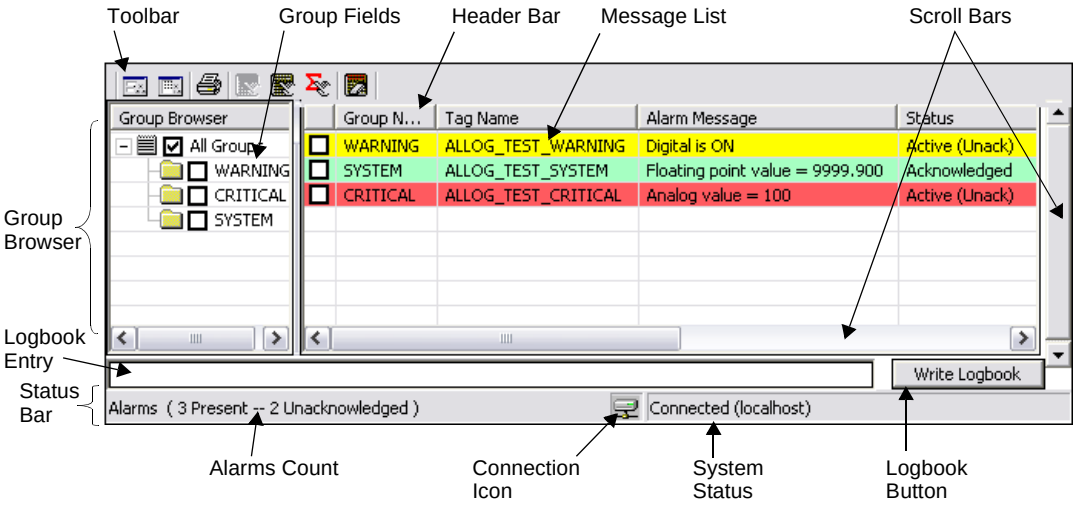
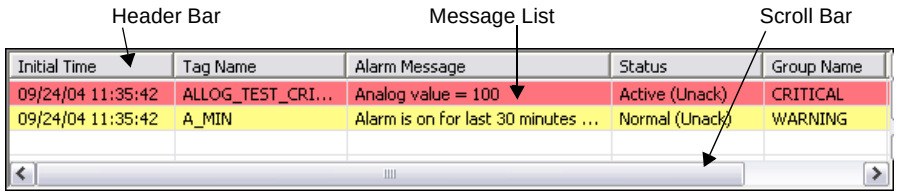


Figure 2-14 Parts of the Alarm Banner Viewer at Run Time



For detailed instructions on using the alarm viewers at run time, see the *Client Builder Help*. When all selections are complete, click **Apply** and then **OK**.

TROUBLESHOOTING

If the Alarm task is not working, please check the following steps. If you used one of the starter applications as a basis for your application, these steps are preconfigured for you.

Verify Alarm Server Task Setup

The Alarm Server task and the associated alarm server executable file are predetermined in the FactoryLink software as part of the System Configuration Information.

The Alarm Server manages the alarm output using the Alarm Server task. In response to an alarm condition, the Distributed Alarm Logger task creates a message. The message can be output to the Alarm Viewer through the Alarm Server, one or more databases, a text file, or a printer.

If you used one of the starter applications as a basis for your application, this information is already completed for you. For most applications, you should not change any of the default information.

Accessing

In your server application, open **System > System Configuration > System Configuration Information > Alarm Server**.

Table 2-7 contains the field definitions and default settings for the Alarm Server task.

Field Descriptions

Table 2-7 Alarm Server Task

Field Name	Definition		Default
Domain	Current FactoryLink versions use the Shared domain.		Shared
Task Information	Task Name	Predefined name which cannot be changed	ALARMSRV
	Task Description	Description	Alarm Server
Task Flags	Run at Startup	**R**: Invokes task at FactoryLink startup	Yes
	Create Session Window	**S**: Provides the process with its own tab window and prints messages to the Configuration Explorer Output window.	Optional
	Suppress Online Configuration	**O**: Suppress online updates for this process	Optional
	Suppress Task Hibernation	**H**: Not applicable	Not applicable
Flag String Value	Input box	Displays value code of selected Task Flags **F**: Puts task in the foreground at startup	Yes
	Edit Flags Directly	If selected, allows user input of string values to input box.	Not selected
Task Options	Start Order	Specifies the run-time rank for invoking the task when FactoryLink is started	1
	Start Priority	Operating system processing priority	201
Task Executable	Executable File	Path and name of file which executes this task	bin/alarmsvr
	Program Arguments	See “Program Arguments” on page 52.	None

Verify the Distributed Alarm Logger Task

The default options for the Distributed Alarm Logger task are preconfigured in the FactoryLink software. The task name AL_LOG is standard and cannot be changed. Other options can be modified by the designer. If you used one of the starter applications as a basis for your application, this information is already completed for you. For most applications, you do not need to change the default information.

Accessing

In your server application, open **System > System Configuration > System Configuration Information > Distributed Alarm Logger** in form view.

Field Descriptions

Table 2-8 System Settings for Distributed Alarm Logger Task

Field Name	Definition		Default
Domain	Current FactoryLink versions use the Shared domain. The User domain is still supported.		Shared
Task Information	Task Name	Predefined name which cannot be changed	AL_LOG
	Task Description	Description	Distributed Alarm Logger task
Task Flags	Run at Startup	**R**: Invokes task at FactoryLink startup	Yes
	Create Session Window	**S**: Provides the process with its own tab window and prints messages to the Configuration Explorer Output window.	Optional
	Suppress Online Configuration	**O**: Suppress online updates for this process	Optional
	Suppress Task Hibernation	**H**: Not applicable for alarm functions	Not applicable
Flag String Value	Input box	Displays value code of selected Task Flags. **F**: Puts task in the foreground at startup. **A**: Suppress printing of return-to-normal messages for toggle type fields	FR
	Edit Flags Directly	If selected, allows user input of string values to input box.	Not checked
Task Options	Start Order	Specifies the run-time rank for invoking the task when FactoryLink is started	2
	Start Priority	Operating system processing priority	201
Task Executable	Executable File	The path and name of file which executes this task	bin/al_log
	Program Arguments	See “Program Arguments” on page 52.	None

Verify the Distributed Alarm Server Table

The Distributed Alarm Server table supplies the mailbox and trigger information for the Alarm Server and the Distributed Alarm Logger tasks to communicate.

If you used one of the starter applications as a basis for your application, this information is already completed for you. For most applications, you do not need to change the default information.

Accessing

In your server application, open **Alarms > Distributed Alarm Server > Distributed Alarm Server** in form view.

Field Descriptions

These tags are required for communication. The time interval for the poll trigger tag can be modified using the Interval Timer table. The alarm server table tags are:

Send Mailbox — Storage of communications retrieved by the Distributed Alarm Logger task

- Default Entry: ALARMSRV_SNDMBX
- Valid Data Type: Mailbox
- Valid Entry: Up to 16 alphanumeric characters

Receive Mailbox — Storage of communications from the Distributed Alarm Logger task

- Default Entry: ALARMSRV_RCVMBX
- Valid Data Type: Mailbox
- Valid Entry: Up to 16 alphanumeric characters

Poll Trigger — Polling frequency of the communications between the Alarms Server and the Distributed Alarm Logger

- Default Entry: ALARMSRV_POLL
- Valid Data Type: Digital
- Valid Entry: tag name

Verify Polling Trigger Timer Setup

The polling frequency for the Distributed Alarms Server is preconfigured in the Interval Timer Information table. The default is a one second timer named ALARMSRV_POLL. If you used one of the starter applications as a basis for your application, this information is already completed for you.

PROGRAM ARGUMENTS

Argument	Description
-A	Disables the “Return-to-Normal” message for digital alarms.
-D<#>	Set debug log level for Run-Time Manager output window. (# = 1 to 9)
-F	Freezes initial text display of alarms configured with %s (C-style) variables.
-G	Ignore remote log settings.
-H<#>	Set historian time-out parameter. (# = 5 to 30 seconds)
-I	Leave Node ID embedded in sequence for logging.
-L	Enables logging of debug information to a log file.
-M<#>	Set maximum number of records in Alarm log text file. (# = 1 to 1000)
-O	Set “log once” mode.
-Q<#>	Set warning limit for historian maximum number of outstanding responses.
-S or -s	Sleep before re-entering DTP wait. This would be used when a lot of alarms which change often are configured but do not result in very many alarms.
-V#	Verbose level increases from 1 through 9.
-W	Warm start; use/maintain a Persistence file of alarms.

ERROR MESSAGES

If the Distributed Alarm Logger task encounters any problems during run time, an error message is written to a log file in addition to displaying on the Run-Time Manager screen. The log file resides in the following directory:

{FLAPP}/{FLNAME}/{FLDOMAIN}/log

Alarm Logging Messages

Error Messages	Cause and Action
Cannot connect to SMTP server	Cause: The server may be offline. Action: If the problem persists, check the e-mail agent SMTP configuration and restart the alarm logger task.
Cannot connect to POP3 server	Cause: The server may be offline. Action: If the problem persists, check the e-mail agent POP3 configuration and restart the alarm logger task.
Configuration could not be loaded	Cause: The configuration data could not be loaded for the e-mail agent. The agent will terminate. Action: Check the e-mail agent's configuration. Make sure the ag.ct file was generated (use CTGEN). Display the CT file to verify if any records were generated.
Deadband tag has a different type at Unique Alarm ID number	Cause: Alarm tag and limit tag are different types. Action: Correct the alarm tag or limit tag type. They must agree.
Duplicate Unique Alarm ID *number* used in CT file	Cause: The program detected a duplicate Alarm ID number in the software. This check is only performed when a **-d** or **-l** parameter is specified in the program arguments. Action: Remove the duplicate from the Alarm Logging definition.
Error (%d) on set change flag	Cause: An internal call to set a change flag on a tag has failed. Action: Report the error to your support representative.
Empty CT file...	Cause: No tables are found in the CT file. Action: The configuration is probably invalid. Fill in the information and define an alarm.

Error Messages	Cause and Action
Error locating a child with Unique Alarm ID *number*	Cause: The program did not find an Alarm ID number used to define a child. This normally means a corrupted CT file. Action: Run `ctgen -r`.
Error locating a parent *Unique Alarm ID* for alarm '*tag name*'	Cause: The program is not able to find an Alarm ID number used to define a parent. Action: Correct the Parent Alarm ID number for the listed child.
Error locating group '%s'	Cause: The Alarm Logging task failed while reading CT files. Action: Ensure the application is restored properly and converted (if necessary). Report the error to Customer Support.
Error on change read	Cause: The kernel returned an error when reading a message from a mailbox. Action: Check the CT files or run `ctgen -r`.
Error reading *type* tag	Cause: The kernel returned an error while reading a tag. Action: Check the CT files or run `ctgen -r`.
Error reading CT index	Cause: An error occurred while reading the CT index. Action: Run `ctgen -r` and check the al_log.ctg file for corruption.
Error reading from CT *table* table	Cause: An error occurred on reading the CT record. Action: Run `ctgen -r` and check the al_log.ctg file for corruption.
Error retrieving message from *type* mailbox	Cause: The kernel returned an error upon retrieving a message from a mailbox. Action: Check the CT files or run `ctgen -r`.
LAN send Mailbox is not empty. Please check FLLAN.	Cause: The FLLAN task did not read messages from the mailbox. Action: Check the Run-Time Manager screen to ensure that FLLANSND and FLLANRCV are running.
Limit tag has a different type at Unique Alarm ID *number*	Cause: Alarm tag and limit tag are different types. Action: Correct the alarm tag or limit tag type. They must agree.

Error Messages	Cause and Action
More *object* than configured!	Cause: More Alarms, Children, or Logbook entries exist than the number specified in the Distributed Alarm Logger Setup table. Action: Increase maximum active alarms in general set up control for al_log.
No CT file found...	Cause: The al_log.ct file was not present. Action: Check the tables for the domain selection and the ctlist file for typing errors.
No database information for logging...	Cause: Logging is requested for *xxx* group, but no set up information is entered. Action: Fill out the Alarm Archive Control table.
No Setup information specified	Cause: Parts of the Distributed Alarm Logger Setup table, which are necessary for startup to occur, are not configured. Action: Complete the general set up.
Not enough memory	Cause: The software was not able to allocate sufficient memory to load all data into memory. Action: You have insufficient memory in your system and must add more memory in order for the program to run.
No valid printer device on 'Print Active List'	Cause: The software is not able to print the active alarm list because no valid device was entered in the Distributed Alarm Logger Setup table. Action: Verify printer device is defined in printer spooler. Consult the printer troubleshooting manual if trouble persists.
Programming Error *number*. Contact Customer Support!	Cause: Internal error message Action: Report the error to your support representative.
Received less bytes than expected	Cause: The FLLAN task is not able to transmit the entire message. Action: Increase the MAXLEN parameter in the local group file.
Sequence error in receiving %s from node %d	Cause: The Alarm Logging task received an inappropriate message. Action: Ensure the physical network connections are correct. Report the error to your support representative.

Error Messages	Cause and Action
Severe error: No memory available for action	Cause: The software is not able to allocate sufficient memory during run time. Action: You have insufficient memory in your system and must add more in order for the program to run.
SQL Error *number, 'message'*	Cause: Historian returned an error and message on an action from the Distributed Alarm Logger task. Action: See the historian error codes on page 291.
Timer task is not running!	Cause: The Timer task was not running before the Logging task was started. All internal time-stampings are retrieved from the Timer. Action: Start the Timer task or, if it is already running, change the sequence to ensure the Timer starts before Alarm Logging.
Unable to insert DTP handler for %s tag	Cause: An internal tag processing call failed. Action: Report the error to your technical support representative.
Unable to open file for active list	Cause: The software cannot open and write to the {FLAPP}/alarms.txt file. The file may be open in another application or you do not have rights to open the file. Action: May not have sufficient rights or enough disk space. Run a CHKDSK command to ensure enough disk space is available.
Unable to open Persistence file '%s'	Cause: The Alarm Logging task is unable to open the alarm persistence file to save data. Action: Check the drive the application is on to ensure enough space is available. Ensure information can be written to the drive. If disk space is low, make space available.
Unable to write to Persistence file '%s'	Cause: The Alarm Logging task is unable to write record(s) to the opened alarm persistence file. Action: Check the drive the application is on to ensure enough space is available. Ensure information can be written to the drive. If disk space is low, make space available.
Warning: More than one line of *table* information	Cause: On tables where only one line is read, more lines are detected. The program will continue but could work on the wrong parameters. Action: Remove the additional lines.

Alarm Viewer Messages

Error Message	Cause and Action
CT file *ctname* not found	Cause: The al_view.ct file was not present. Action: Check the tables for the domain selection and the ctlist file for typing errors.
Dimensions of array too short, view window *number*	Cause: The array dimensions and the number of lines do not match. The array is too small. Action: Correct the number of lines or redefine the array.
Duplicate Unique Alarm ID *number* used in CT file	Cause: The program detected a duplicate Alarm ID number in the software. This check is only performed when a **-d** or **-l** parameter is specified in the program arguments. Action: Remove the duplicate Alarm ID.
Error clearing chg flag *type error*	Cause: The kernel returned an error while clearing a change flag. Action: Check CT files or run `ctgen -r`.
Empty CT file table *'ctname'*	Cause: No tables are found in the CT file. Action: Check Alarm Logging set up for table names.
Error *error* on change read	Cause: The kernel returned an error while reading a changed tag. Action: Check CT files or run `ctgen -r`.
Error operation on *operation* mbx *number*	Cause: The kernel returned an error on a mailbox operation. Action: Check the CT files or run `ctgen -r`.
Error reading CT index...	Cause: An error occurred on reading the CT index. Action: Run `ctgen -r` and check the al_view.ctg file for corruption.
Error reading from CT file.	Cause: An error occurred on reading the CT record. Action: Run `ctgen -r` and check the al_view.ctg file for corruption.
Error reading type *tag error*	Cause: The kernel returned an error while reading a tag. Action: Check CT files or run `ctgen -r`.
Error writing type *tag error*	Cause: The kernel returned an error while writing a tag. Action: Check CT files or run `ctgen -r`.

Error Message	Cause and Action
***Method data* for filtering does not exist**	Cause: The user-defined data does not match any of the available alarms or groups. Action: You are trying to filter or sort a group that is not present. Adjust your sort method.
No Setup information specified	Cause: Parts of the Alarm Viewer table necessary to operate are not configured. Action: Complete the parts of the Alarm Viewer table.
Not enough memory	Cause: The software is not able to allocate sufficient memory to load all data into memory. Action: Your system has insufficient memory to run the program. Add more memory.
Severe error, no memory available for *action*	Cause: The software is not able to allocate sufficient memory during run time. Action: Add more memory.
Unable to load key file	Cause: One of the key files used for translation is missing. Action: Reinstall FactoryLink.
Unknown alarm group *group*	Cause: The program is not able to find the Alarm ID number used to define a child. Action: Check to see if your CT file is corrupted.
Unknown CT type...	Cause: An unknown type of CT in the CT file is displayed. Action: Run `ctgen -r` and check the al_view.ctg and al_info.ctg files for corruption.
Warning: More than one line of *table* information	Cause: On tables where only one line is read, more lines are detected. The program will continue but could work on the wrong parameters. Action: Remove the additional lines.
Wrong mailbox message type received *number*	Cause: This error should not occur. Action: Report the error to your technical support representative.

Chapter 3 . . . .
Batch Recipe

The Batch Recipe task transfers sets of predefined values, sometimes called recipes, between binary disk files and selected tags in the real-time database. In the real-time database, a batch recipe is a collection of tags grouped together for some purpose. These tags can contain internally-generated or operator-entered values.

Depending upon the type of recipes that you need, you might find it preferable to use one of the relational databases to create and store your recipes.

OPERATING PRINCIPLES

You can perform the following functions with Batch Recipe:

- Define up to 8,000 different recipe templates, each associated with a virtually unlimited number of files
- Store batch recipes in disk files so the total number of different recipes stored on a system is limited only by available disk space
- Store each batch recipe file under a standard file name
- Specify up to 8,000 tags for one batch recipe template
- Use with any of these data types: digital, analog, long analog, floating-point, and message

You can configure Batch Recipe for use in many diverse applications. For example, a program can use a graphic display for the entry of application values and write these values to an external device using an external device interface task. Batch Recipe can save these tag values in a recipe so the program can then read the values from the batch recipe file.

Sample applications that use a single batch recipe template include:

- Producing a particular line of paint. You can use multiple recipe files using the same recipe template to set various hues or colors of the paint being produced.
- Setting up external devices with different recipes for days of the week, end of the month, and other schedules.
- Setting up an environment for a testing procedure with different sets of testing parameters.

You can use batch recipes in conjunction with any FactoryLink task because each FactoryLink task communicates with other tasks through the real-time database.

Batch Recipe executes as a background task. The task does not require operator intervention at run time unless you design the application to require it.

You can configure Batch Recipe to be triggered by events, timers, or operator commands, such as:

- An external device read operation
- A Math & Logic calculation
- An activity from another station on a network
- Input from the operator using a keyboard or pointing device

Perform the following steps to configure a typical Batch Recipe application:

1. If the recipes are to be edited and/or viewed, create and animate a mimic with fields to enter or display tag values in the recipe.
2. Complete a Recipe table containing the tags to be used in the recipe.
3. Complete the protocol module Read/Write table for an external device if you want to connect the tags in the recipe with addresses in the device.

Monitor the Run-Time Manager screen to determine the status of Batch Recipe at run time.

> **Note:** When performing a platform-dependent FLSAVE, FactoryLink saves recipe files; however, when performing a platform-independent or multiplatform FLSAVE, FactoryLink does not save recipe files.

RECIPE CONTROL TABLE

Accessing

In your server application, open **Recipe > Recipe > Recipe Control.**

Field Descriptions

Recipe Name Unique name of the recipe template to be defined or modified.

Valid Entry: up to 16 alphanumeric characters

Read Trigger — Tag that initiates a read operation. When Recipe detects this tag is forced to 1 (ON), the task reads the values from the disk file specified in the File Spec. and File Spec. Variable fields and transfers them to the tags specified in the Recipe Information table.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

Save Trigger — Tag that initiates a write operation. When the value of this tag changes, Recipe collects the current values of the tags specified in the Recipe Information table and writes them to the binary disk file specified in the File Spec. and File Spec. Variable fields.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

File Spec. — Variable specifier that uses the value of the File Spec. Variable tag as the file path and name.

Sample Path—if you specify the Path File Name **DISK:/RECIPE/PAINT%03d.RCP**, and define the File Spec. Variable tag as an analog data type, value 23, the system generates the following filename: **DISK:/recipe/paint023.rcp**

Because the default path is **/FLAPP/FLNAME/FLDOMAIN/FLUSER/RCP**, the **File Spec.** of **PAINT/%s.RCP** and the **File Spec. Variable** containing a value of 'red' generate the following filename:

/FLAPP/FLNAME/FLDOMAIN/FLUSER/rcp/paint/red.rcp

If the File Spec. Variable is absent, the filename is generated from the **File Spec.** and the default path name **/FLAPP/FLNAME/FLDOMAIN/FLUSER/RCP**. If the **File Spec.** is absent, it defaults to **%s**,**%d**,**%1d**, or **%-8.3f** as appropriate for the File Spec. Variable data type.

For more information on format specifiers, see Appendix, "Format Specifiers."

File Spec. Variable — Name of a tag whose value is used with the entry in the **File Spec.** field to form the file/path name for a binary disk file that contains a specific recipe.

Valid Entry: tag name

Max. Msg. Length — The maximum number of characters in a message.

Valid Entry: 1 to 255

Forced Write

Indicator of whether all change-status flags on the tags in the recipe are to be set to 1 (ON) when a read operation occurs. Batch Recipe can set change-status flags in a read operation for all tags specified in a recipe rather than only for those tags whose values have actually changed since the last read operation. This can be one of the following:

YES	Sets change-status flags for all tags when read regardless of their actual change status
NO	Causes change-status flags to be set only for tags whose values have changed since the last read. This is the default.

Completion Trigger

Name of a tag whose value is forced to 1 (ON) by the Recipe task when a read or write operation is completed. The value of the Completion Trigger tag is 0 (OFF) when the program begins loading a new Batch Recipe file from the specified drive to the real-time database, and is forced to 1 (ON) when the file finishes loading.

Completion Triggers can be used in multiple operations. For example, the Completion Trigger can initiate a write of a recipe to an external device and it can initiate a message to the operator on a mimic. The Math & Logic task can check the trigger to determine successful reading or writing of the recipe.

Valid Entry: tag name
Valid Data Type: digital

Completion Status

Name of a tag set to 0 (OFF) by the Recipe task when the last recipe read or write is completed without an error or set to 1 (ON) when the last recipe read or write is completed with an error.

Valid Entry: tag name

RECIPE INFORMATION TABLE

Accessing

In your server application, open **Recipe > Recipe > Recipe Control > "your recipe name" > Recipe Information**.

Field Description

Tag Name — Names of tags to be read or written from the recipe file.

Valid Entry: tag name

Valid Data Type: digital, analog, longana, float, or message

SAMPLE BATCH RECIPE

This section demonstrates how to configure Batch Recipe for the production of cookies and cereal using the same recipe template. You create a recipe at run time by entering values for temperature, cook time, flour, water, and sugar on the recipe mimic. Batch Recipe accesses these values for communication to an external device that controls production. You can also create other recipes from the same recipe template.

Perform the following steps to configure this sample recipe template:

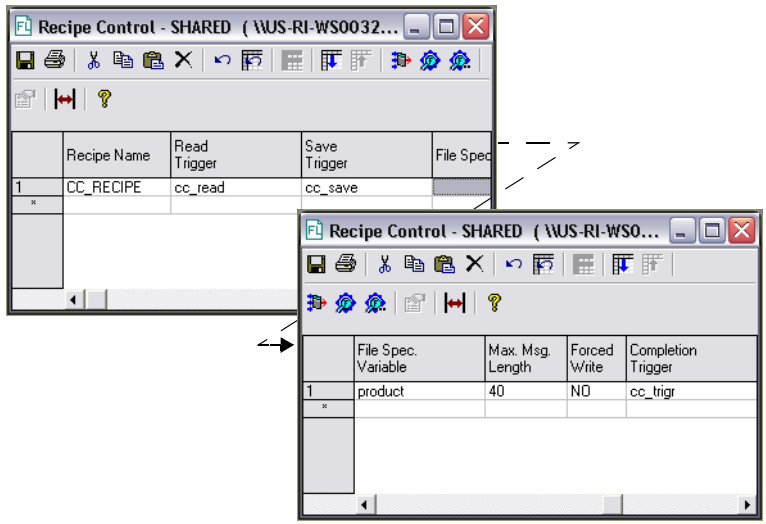
1. Configure a batch recipe template.
2. Link recipe input values to an external device.
3. Draw and animate a mimic for the operator to use to create and edit recipes at run time.

Configuring Batch Recipe Template

Configure a batch recipe template to create multiple batch recipes without completing a new recipe table for each recipe. In this sample application, the template creates a new recipe and opens an existing one.

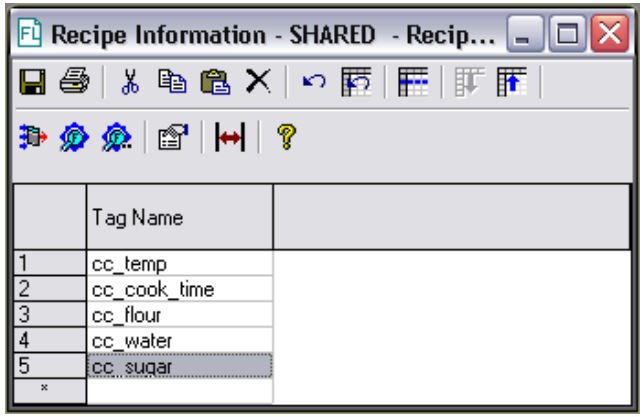
Perform the following steps to complete the Recipe table:

1 In your server application, open **Recipe > Recipe > Recipe Control**.



2 Enter the tags as shown in the tables above and save the information when the Recipe Control table is complete.

3 Open **Recipe > Recipe > Recipe Control > CC_RECIPE > Recipe Information** and enter the names of the tags to be used in the recipe template as shown below.



The entries in this table specify the tags (**cc_temp, cc_cook_time, cc_flour, cc_water**, and **cc_sugar**) whose values you enter on the RECIPE display at run time.

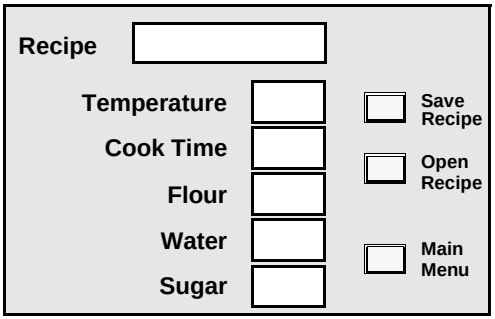
4 Save the information.

Linking Recipe Input Values to an External Device

Link the values to registers in the external device if you want the input values of the recipes to change the settings in an external device. In this example, you would configure a PLC Write table with the same tags as those that are used in the recipe. Your PLC ladder logic would need to written to correspond to the settings in the recipe. You can also read values from an external device and save those in a recipe to be used at a later time.

Drawing and Animating a Mimic

Draw and animate a mimic called **RECIPE** using the Client Builder.



Link the tags you created in the recipe table to the animated fields in the mimic.

At run time, you create a recipe the first time you open this mimic by entering a Recipe name and entering values for Temperature, Cook Time, Flour, Water, and Sugar. To save the recipe, the operator clicks **Save Recipe** and then the Batch Recipe:

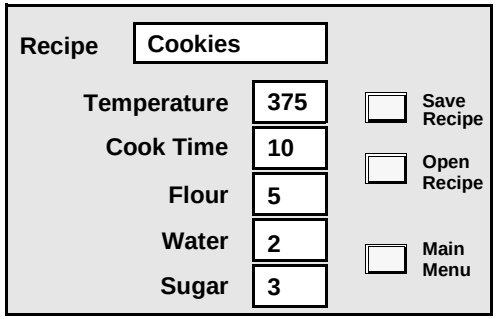
- Writes the values you just entered to the tags **cc_temp**, **cc_cook_time**, **cc_flour**, **cc_water**, and **cc_sugar** you defined in the Recipe Information table when you configured the task.
- Creates a recipe file (with a .RCP extension) and writes the values of these tags in binary form to this file.

When you wish to retrieve the recipe later for display on the screen, open the RECIPE mimic, type the recipe name, and click **Open Recipe**. Batch Recipe collects the binary values from the disk file, writes them to the tags, and the Graphics task displays them on the mimic.

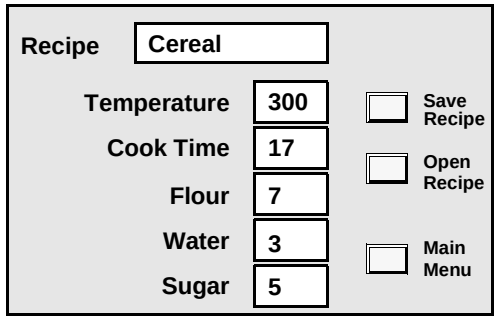
Run-time Example

1 In the System Configuration table, be sure the recipe task is configured and has the **R** flag to start.

2 To create a recipe for Cookies, enter the values shown in the following figure and click **Save** Recipe to write these values to the tags specified and store the recipe under the file name **Cookies**.rcp.



3 To create a recipe for making cereal, type **Cereal** in the **Recipe Name** field and enter the following values for each variable.



4 Click **Save Recipe**. Batch Recipe writes the values for the cereal recipe to the specified tag in the real-time database and stores them on disk in the binary file named CEREAL.RCP.

5 To open the recipe for Cookies, type **Cookies** in the **Recipe Name** field and click **Open Recipe** to recall the recipe for cookies. Batch Recipe reads the values for each of the variables from the binary disk file, deposits them in the real-time database, and displays them on the screen.

PROGRAM ARGUMENTS

Argument	Description
-L or -l -	Enables logging of debug information to a log file.
-V	Does the same as argument –L.

ERROR MESSAGES

Error Message	Cause and Action
Can't create recipe file *filename*	Cause: The file name or pathname may be invalid or the disk may be full. Action: Specify the Path File Name correctly. If the Path File Name is correct, ensure the directory exists. Run **CHKDSK** or any disk diagnostic program to determine whether the disk is full. Delete unnecessary files if the disk is full or nearly full. Consider making this part of your ongoing maintenance effort.
Can't find recipe *filename*	Cause: The task cannot locate the specified recipe file. Action: Either the recipe file does not exist or the name of the file was entered incorrectly.
Error reading CT header	Cause: Either the RECIPE.CT file is corrupt or the .CT script file (**/FLINK/CTG/RECIPE.CTG**) and the FactoryLink Run-Time version are not the same version. Action: Delete the .CT file and then restart the application to rebuild the file, or execute **CTGEN RECIPE.CT -V3**.
Error reading CT index	Cause: The file **RECIPE.CT** file has been damaged. Action: Rebuild the file by deleting the RECIPE.CT file and restarting the application, or execute **CTGEN RECIPE.CT -V3**.
Error reading CT record	Cause: Either the RECIPE.CT file is corrupt, or the .CT script file (**/FLINK/CTG/RECIPE.CTG**) and the FactoryLink Run-Time version are not the same version. Action: Delete the .CT file and then restart the application to rebuild the file, or execute **CTGEN RECIPE.CT -V3**.
Error reading recipe *filename*	Cause: Either the file is corrupt or the disk is damaged. Action: Inspect the file and run a disk diagnostic program to determine if the disk is corrupt. If it is, recreate the file from scratch or from the backup disk or tape.

Error Message	Cause and Action
Error writing recipe *filename*	Cause: The specified device is either full or corrupt. Action: Run **CHKDSK** or any disk diagnostic program to determine whether the disk is full. Delete unnecessary files if the disk is full or almost full.
Invalid CT header size	Cause: Either the RECIPE.CT file is corrupt or the .CT script file (**/FLINK/CTG/RECIPE.CTG**) and the FactoryLink Run-Time version are not the same version. Action: Delete the .CT file and then restart the application to rebuild the file, or execute **CTGEN RECIPE.CT -V3**.
Invalid CT record size	Cause: Either the RECIPE.CT file is corrupt, or the .CT script file (**/FLINK/CTG/RECIPE.CTG**) and the FactoryLink Run-Time Manager are not the same version. Action: Delete the .CT file and then restart the application to rebuild the file, or execute **CTGEN RECIPE.CT -V3**.
Invalid tag	Cause: The program detects an invalid tag or the tag does not exist in the real-time database. Either data in the real-time database is corrupt or the Run-Time Manager was started without having run **CTGEN**. Action: Run **CTGEN**.
No recipes defined	Cause: No recipes are defined in the Recipe Information table. Action: Remove the R flag for the Recipe task from the System Configuration table or define at least one recipe.
No tables configured for this task	Cause: Either you did not configure the Recipe table or the following files could be missing or damaged: - RECIPE.CDB database - RECIPE.MDX database - /FLAPP/CT/RECIPE.CT - .CT script /FLINK/CTGEN/RECIPE.CTG FLNEW or CTGEN may not have run correctly. Action: Delete the .CT file. Run **CTGEN** and try again.
Not enough RAM to load CTs	Cause: The task does not have enough RAM allocated to load the configuration tables. Action: Shut down any unnecessary tasks. Increase the system RAM size if this does not help.

Error Message	Cause and Action
No triggers configured for this task	Cause: No read or write triggers are defined in the Recipe table. Action: Define read and write triggers in the Recipe table.
Recipe doesn't match CT *filename*	Cause: The specified recipe file does not match the Recipe Configuration table. The number of tag names for each data type and the maximum string space may not match. Action: Reenter the recipe file. If the file is correctly specified, inspect the recipe file in the configuration table.
Recipe is too big	Cause: The total tags in a recipe will not fit in a 10000-byte buffer. Action: Create a smaller recipe, decrease the size of the message strings, or switch from longana to analog data type.

Chapter 4 . . . .

Client Builder

A FactoryLink application consists of a client project that is configured in the Client Builder and a server application that is configured in the Configuration Explorer. In the Client Builder environment, you create and configure the graphical user interfaces for your FactoryLink application to graphically represent your industrial processes. Client Builder also provides the run-time environment for interacting with those interfaces.

Although most of the procedures in this guide are done using the Configuration Explorer, some procedures reference the Client Builder. You can access the Client Builder by double-clicking the Client Builder icon on your desktop. For detailed information, procedures, and program arguments to use the Client Builder, see the *Client Builder Help*.

- **4 | CLIENT BUILDER**
-
-
-

Chapter 5 . . . .

Database Browser

The Database Browser task works in conjunction with the historian tasks to allow a server application to access data in a relational database through a browse window. This method of browsing is more flexible and powerful than using the Database Browser Control, but requires more configuration effort.

Database Browser offers the following features:

- Allows relational data in a relational database to be manipulated from within FactoryLink
- Allows an application to send and retrieve data to and from all external database tables, including those created outside of FactoryLink
- Allows you to define tags referenced by Database Browser in arrays as well as individually

> **Note:** If you are starting a new application, see the *Fundamentals Guide* for a discussion on various browsing options. The Database Browser Control may suit your needs and require less time to configure. PowerSQL has all the functionality of the Database Browser task and offers even more power and flexibility with about the same configuration effort as the Database Browser task. For detailed information and procedures to use the Database Browser Control, see the *Client Builder Help*. For detailed information and procedures to use PowerSQL, see page 411.

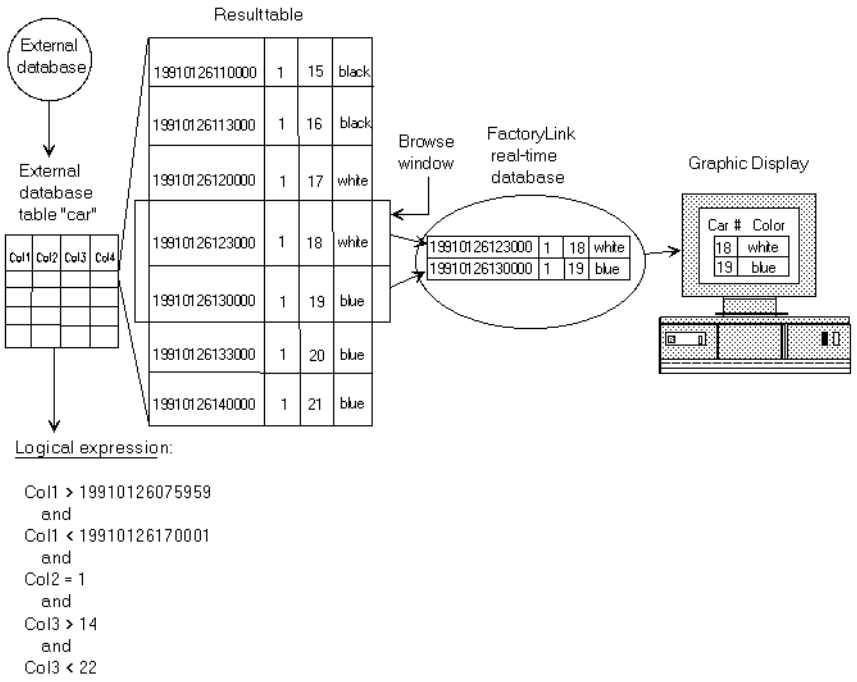
Operating Principles

Database Browser is a historian-client task that communicates with a historian through mailbox tags to send and receive historical information stored in an external database.

Database Browser accesses data in a relational database by selecting the data specified in a configuration table and placing it in a temporary table called a result table. The task views and modifies the data in the result table through a browse window. A browse window is a sliding window that maps data between the relational database and the real-time database. The browse window views selected portions of the result table.

For example, if a mimic is used to display the browse window, it can display as many rows of data from the result table as there are tags in the two-dimensional tag array. If there are more rows in the result table than in the browse window, the operator can scroll through the result table and see each row of it in the browse window.

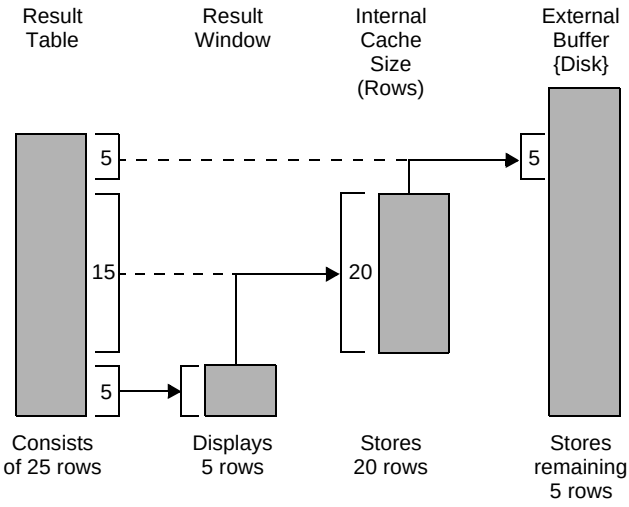
The relationships among the external database, the result table, the browse window, the real-time database, and the mimic are displayed below.



Database Browser can read from and write to an entire array of tags in one operation.

An internal buffer stores the rows of the result table in RAM. An external buffer stores the overflow of rows from the internal buffer on disk. This allows the operator to scroll back up through the result table. Figure 5-1 shows the buffers.

Figure 5-1 Buffers Used in Database Browser



In this example, as the operator scrolls through the result table, the rows of the result table flow into the internal buffer to be stored in memory. Because, in this case, the result table consists of 25 rows and the internal buffer can store only 20 rows, when the internal buffer is full, the excess rows in the internal buffer flow into the external buffer to be stored on disk.

USE OF LOGICAL EXPRESSIONS

You use logical expressions to specify the data in a relational database to view or modify. For the purposes of the Database Browser task, a logical expression is a command containing a standard Structured Query Language (SQL) WHERE clause.

To select data from a database table, a logical expression works in conjunction with the table's column name and logical operators to form an SQL WHERE clause. The WHERE clause specifies which rows in a database table to place in the result table.

> **Note:** You must know how to write a standard SQL statement to configure the Database Browser task. For additional information, see any SQL guide or the user manual for the relational database.

To make a logical expression flexible at run time, use the name of a tag whose value is a WHERE clause. If viewing all data from a column in a relational database table, you do not need to specify a logical expression.

The following table represents part of a sample database table, CAR.

TRANDATE	CONVEYOR	CARNUM	COLOR
19910126093000	1	12	red
19910126100000	1	13	silver
19910126103000	1	14	black
19910126110000	1	15	black
19910126113000	1	16	black
19910126120000	1	17	white
19910126123000	1	18	white
19910126130000	1	19	blue
19910126133000	1	20	blue
19910126140000	1	21	blue
19910126143000	1	22	brown

A sample WHERE clause referencing the previous table CAR is

```
TRANDATE > '19910126075959' AND TRANDATE < '19910126170001' AND
CONVEYOR = 1 AND CARNUM > 14 AND   CARNUM <21
```

In this example, the WHERE clause requests:

```
Which colors cars 15 through 20 on conveyor 1 were painted between
8:00 A.M. and 5:00 P.M. on January 26, 1991
```

From this WHERE clause, the relational database places the following values in a result table.

19910126110000	1	15	black
19910126113000	1	16	black
19910126120000	1	17	white
19910126123000	1	18	white
19910126130000	1	19	blue
19910126133000	1	20	blue

If the view size of the browse window is 2, the browse window writes the values of the tags in two rows to the real-time database, where other FactoryLink tasks can read it and write to it, and an operator can view the data on a mimic.

The Database Browser task performs four operations:

- Select—Uses an SQL select statement to select and retrieve data from a relational database to be displayed in a result table.
- Update—Updates the data in the result table and external database. The Database Browser task can perform two types of update operations:
  - Positional—Updates the current row (row at which the cursor is currently pointing) of data displayed in the browse window.
  - Logical—Updates the data described by the logical expression.
- Insert—When an update operation cannot be performed because the row to be updated does not exist, inserts a new row of data in the result table.
- Delete—Deletes a row from the result table and external database. The Database Browser task can perform two types of delete operations:
  - Positional—Deletes the current row (row at which the cursor is currently pointing) of data displayed in the browse window.
  - Logical—Deletes the data described by the logical expression.

DATABASE BROWSER CONTROL TABLE

Accessing

In your server application, open **Data Logging > Database Browser > Database Browser Control.**

> **Note:** If you are using Client Builder, it is recommended that you use the Shared domain. For ECS Graphics, use the User domain.

Field Descriptions

Browse Name — Specifies the developer-assigned name of the browse window being defined or modified.

Valid Entry: up to 15 alphanumeric characters

Select Trigger — Tag that triggers a select operation. A select operation selects specific data from a relational database table based upon information specified in the Database Browser Information table and places it in a result table for you to view or manipulate.

Valid Entry: tag name

Valid Data Type: digital, analog, longana, float, message

Update Trigger	Tag that triggers an update operation. The Database Browser task performs a positional update if you defined a select trigger. When the value of this tag changes during a positional update, the Database Browser task reads the values in the active row (the value of the current row tag) and updates the values in that row of the result table and external database. For a positional update to work, the database table must have a unique constraint configured for it; that is, a unique index must exist for the database table. This can be configured in Database Schema Creation or executed externally to FactoryLink whenever the database table is created. Consult the RDBMS users manual if you need to create a unique index on a database table that already exists. The task performs a logical update if you have not defined a select trigger to select specific data. During a logical update, the Database Browser task reads the values in the first row of the browse window and uses the logical expression defined in the Database Browser Information table to update the values in the external database. Valid Entry: tag name Valid Data Type: digital, analog, longana, float, message
Delete Trigger	Tag that triggers a delete operation. The Database Browser task performs a positional delete if you defined a select trigger. The Database Browser task deletes the active row in the browse window from the result table and external database when the value of this tag changes during a positional delete. For a positional delete to work, the database table must have a unique constraint configured for it; that is, a unique index must exist for the database table. This can be configured in Database Schema Creation or executed externally to FactoryLink whenever the database table is created. Refer to Database Logging for more information on configuring the Database Schema Creation table. Consult the RDBMS user's manual if you need to create a unique index on the database table that already exists. The Database Browser task performs a logical delete if you have not defined a select trigger. The rows are deleted in the relational database indicated by the logical expression during a logical delete. Valid Entry: tag name Valid Data Type: digital, analog, longana, float, message
Move Trigger	(Requires use of the Select Trigger.) Tag that moves the active row up or down the indicated number of rows. The window scrolls the remaining number of records if the active row reaches the first or last record in the

browse window. For example, if the value of the Move Trigger tag is -3 and the active row is positioned on the first row displayed in the browse window, the data in the window scrolls down three rows.

Move operations can be performed only on result tables, so they cannot be performed unless you have executed a Select Trigger.

Valid Entry: tag name
Valid Data Type: analog

Position Trigger

(Requires use of the Select Trigger.) Tag that moves the browse window to the specified row in the result table. The specified row is centered in the browse window and becomes the active row. For example, if the value of this tag is 42, the browse table displays row 42 of the result table.

Position operations are performed only on result tables, so they cannot be performed unless you execute a Select Trigger.

Valid Entry: tag name
Valid Data Type: analog

Historian Mailbox

Mailbox tag whose value initiates communication with an external database. The Database Browser task sends requests for information from the relational database to this mailbox tag. The historian task reads this tag and transfers the request to the external database.

Valid Entry: tag name
Valid Data Type: mailbox

Database Table Name

Specifies the Database Alias Name (defined in the historian task) and the name of the table in the relational database the Database Browser task requests information from. Place a "." between the Database Alias Name and the Table Name.

Valid Entry: up to 63 alphanumeric characters

Current Row Tag

Tag whose value indicates the position of the active row of data in a browse window. After the Database Browser task performs a Select, Move, or Position, the Database Browser task writes the value indicated by the position of the active row to this tag.

The Database Browser task performs all update and delete operations on the row indicated by the Current Row Tag tag if you have defined a select trigger.

Valid Entry: tag name
Valid Data Type: analog

Auto Create Record

Indicates whether a new row is to be inserted in a database table if a row cannot be found when an update operation is being attempted. Works with logical updates but not with positional updates.

YES Insert a new row of data.

NO Do not insert any new rows. This is the default.

Browse Table Size (Rows)

Specifies the number of rows in a browse window that can be viewed or modified. The browse window size must be the same size as the tag array specified in the Tag Name field of the Database Browser Information table. All tag arrays specified in the Tag Name field must be the same size. The browse window size must be the same as the size of the smallest tag array if all tag arrays are not the same size. The Browse Table Size (Rows) field also specifies the number of rows of data sent to the Database Browser task each time it requests data from a historian.

Valid Entry: 1 to 50

Internal Buffer Size (Rows)

Specifies the number of rows of data in a result table that can be stored in memory. Use the following guidelines to choose appropriate internal and external buffer sizes.

The Database Browser task operates more quickly if all rows in a result table are stored in the internal buffer as opposed to being stored in the external buffer.

Use a value large enough to contain as many rows as necessary but small enough not to use up too much memory.

If the size of the result table is unknown and if memory allows, we recommend you enter 100; however, if the result table will be shorter, enter a number equalling (n)-(tag array size), where n is the number of rows in the result table.

The overflow is stored in the external buffer if you choose to store only a given number of rows in the internal buffer and the result table grows larger than the internal buffer.

If: the internal buffer can store 25 rows
and the external buffer can store an unlimited number of rows
and the browse window has 5 rows
and the result table contains 50 rows,

then: 5 rows display in the browse window
25 rows are stored in the internal buffer
and 50 rows are stored in the external buffer

Valid Entry: 1 to 9999

External Buffer Size (Rows)

See discussion above in the Internal Buffer Size field description.

Valid Entry: 1 to 9999

Disable Tag

Tag that disables all related browse operations.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, or float

Completion Trigger

Tag whose change-status flags are set by the Database Browser task when any browse operation for this browse window is complete.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

Completion Status

Tag whose value indicates the status of the current operation completed by the Database Browser task or historian. The status displays as a character string if this tag is a message tag; otherwise, it displays as a numeric code. If the tag is digital tag, a 0 (zero) indicates success, and a 1 (one) indicates failure. If the tag is an analog tag, the actual return from the database software (which could be greater than 32767) is translated to an equivalent historian error and returned in the status tag. If the configuration tag is longana or float type, it returns the actual status received from the database. See “Status Codes and Error Messages” on page 88 for the codes and messages that can display in this tag.

You can configure this tag to work in conjunction with output objects in Client Builder to display codes or messages on any mimic. You can also configure Math & Logic to monitor this tag and respond to or ignore errors that occur.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

DATABASE BROWSER INFORMATION TABLE

Accessing

In your server application, open **Data Logging > Database Browser > Database Browser Control > "your browser table" > Database Browser Information.**

Field Descriptions

Tag Name Tag that contains the values from a column of a relational database table. If the Browse Table Size field in the Database Browser Control table is greater than 1, the tag must be an array of Browse Table Size or greater. Ensure all tags entered in the Tag Name field can accommodate Browse Table Size.

Valid Entry: tag name

Valid Data Type: digital, analog, longana, float, message

Logical Operator Part of a WHERE clause that specifies the conditions the Database Browser task uses to select rows from a relational database table. This field works in conjunction with the Column Name and Logical Expression fields (described below) to specify WHERE clauses.

- **AND** Specifies a combination of conditions in a logical expression.
- **OR** Specifies a list of alternate conditions in a logical expression.

  FactoryLink performs a sequential search through the database even if the columns are indexed if you use the OR operator in a logical expression when using the Historian for dBASE IV. This may result in a slower response time if the database is large.
- **NOT** Negates a condition in a logical expression.
- **AND_NOT** Specifies a combination of conditions and negated conditions in a logical expression.
- **OR_NOT** Specifies a list of alternate negated conditions in a logical expression. (See examples in the following table.)

WHERE clause	Description
Col2 = 3 AND Col4 > 4	The Database Browser task selects all rows where Col2 is equal to 3 AND Col4 is greater than 4.
Col3 < 6 OR Col2 >= 19	The Database Browser task selects all rows where Col3 is less than 6 OR Col2 is greater than or equal to 19.
Col4 > 7 AND_NOT Col4 = 20	The Database Browser task selects all rows where Col4 is greater than 7 AND_NOT equal to 20.

Column Name Specifies:

(1) Relational database column name associated with the Tag Name tag. The Column Name field works in conjunction with the Logical Operator and Logical Expression fields to specify WHERE clauses with the following format:

```
([table.]column)
```

where

table — Is the relational database table name. Include table, if the table name is different from the table name specified in the Database.Table Name field in the Database Browser Control table.

column — Is the name of the column within the relational database table. Use the same column name in two rows of a table.

OR

(2) SQL function, such as MAX (col_name) or COUNT (*). The result of this function is written to the tag specified in the Tag Name field. SQL functions are supported only in SELECT statements. SQL functions are not supported in UPDATE statements or by the Historian for dBASE IV.

Valid Entry: alphanumeric string of 1 to 63 characters

Logical Expression Conditional statement that restricts the rows selected, updated, or deleted from a database table. This field works in conjunction with the Column Name and Logical Operator fields to generate the WHERE clause used by the SQL statement.

Note: An embedded variable in Database Browser is a FactoryLink tag that is preceded by a colon. The embedded variable can only be used in the Logical Expression field. The embedded variable can be any FactoryLink tag type except mailbox. If the tag is an array specify the dimension (for example, tag_xyz[2]). The tag in the embedded variable is not detected by Configuration Explorer as a tag, therefore user must define the tag somewhere else in the application such as in Math & Logic.

The conditional statement in a Logical Expression field can consist of relational operators. The following is a list of relational operators that are supported by the dBASE IV historian.

=	is equal to
<	is less than
>	is greater than
<>	is not equal to
<=	is less than or equal to
>=	is greater than or equal to
is not null	is not a null value (for dBASE IV historian TRUE when database column is not all spaces)
between X and Y	defines a range of values where X is the lower limit and Y is the higher limit. This is equal to COLNAME >= X and COLNAME <= Y

If you are not using the dBASE IV historian, refer to the RDBMS SQL Language user's manual for more information.

The WHERE clause is generated by appending the Local Operator, Column Name, and Logical Expression fields in the order displayed in the Database Browser Information table. Punctuation is supplied by the Database Browser to ensure correct SQL syntax. Any embedded variable found in the Logical Expression field is replaced by a ?, which SQL defines as a substitution marker for a value to be supplied at execute time. The value supplied is the tag's value defined by the embedded variable.

The string generated by this is a WHERE condition. If the first word(s) in this string is not an SQL reserved word such as ORDER BY, then the reserved word WHERE is attached to the start of this string. Ensure that any placement of SQL clauses such as ORDER BY and GROUP BY is properly ordered as defined by the SQL language for the targeted database server.

The ORDER BY clause is supported in the dBASE IV historian but only to the extent that the columns listed in the ORDER BY clause must match an index that was created for the database table. The dBASE IV historian does not build any temporary tables to reorder the rows, so be sure the ORDER BY matches an index for the dBASE IV database table. If an ORDER BY clause does not match an index, the dBASE IV historian returns an error.

If you define a Select Trigger in the Database Browser Control table, the WHERE clause is used for the selected statement. If a Select Trigger is not defined, the WHERE clause is used for the either the update operation or delete operation or both.

A Logical Expression can contain one of the following:

(1) Character string of up to 79 characters containing an SQL expression or an SQL clause.

For example, an SQL expression:
OUTLETVAL = 30 and TANKID = 'BLUE001'

For example, an SQL clause:
ORDER BY TANKID

(2) Character string of up to 79 characters representing an SQL expression that contains embedded variables. If the tag is a message tag, the character data in the message tag should not be enclosed in single quotes.

For example:
=:tagTANKID
WHERE tagTANKID is a message tag of
value: *BLUE001*

(3) An embedded message variable only. THis variable must be a message tag. The message tag contains an SQL clause or SQL expression. THe SQL expression cannot contain an embedded variable and any string constants in the SQL expression must be quoted in single quotes.

For example:
:tagSQLExpression
WHERE tagSQLExpression is a message tag
OUTLETVAL = 30 and
TANKID = 'BLUE001'

Note: The result is the same for Options 1 and 3, but Option 3 allows the user to change the SQLExpression tag to a different expression before setting a select, update, or delete trigger, thereby altering the rows selected, updated, or deleted. Option 1 is always static and cannot be changed at run time. Option 2 allows the user to change the value of TANKID but the SQL expression is still the same; just the search criteria for the where clause changed.

The Database Browser substitutes variables with the value of the tag defined in the embedded variable when executing the select, update, or delete SQL statement.

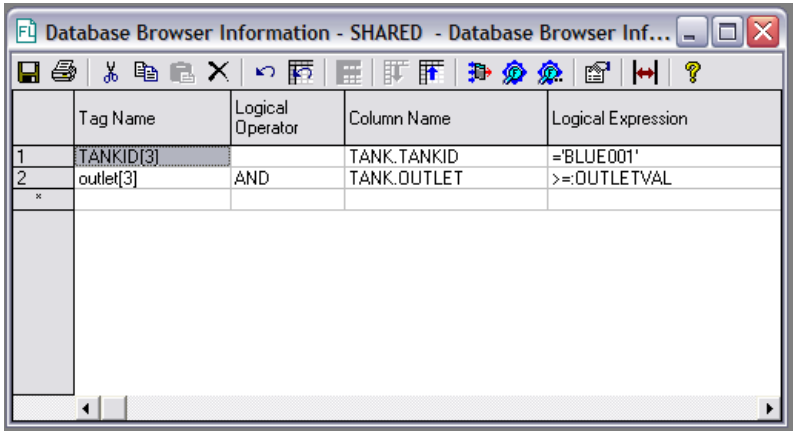
For example: =:tagTANKID

generates the following clause:
WHERE TANKID = ?

TANKID is the value of the Column Name field.

The Database Browser reads the value of the tag tagTANKID from the real-time database and substitutes its value for the ? whenever it executes a select, update, or delete SQL statement.

The table resembles the following sample table when complete.



Because the Select Trigger tag SELTAG1 (defined in the Control table) is digital in this example, the historian returns the two following values to the Database Browser task when the change-status flag for SELTAG1 is set:

- Values where the column named TANKID equals BLUE001
- The column named OUTLET is greater than or equal to the value of the tag OUTLETVAL.

The Database Browser task writes these values to the tags contained in the tag arrays TANKID[3] and OUTLET[3]. These values are then displayed in a browse window.

PROGRAM ARGUMENTS

Argument	Description
-L or -l	Enables logging of debug information to a log file. By default, the Database Browser does not log errors.
-N or -n	Notifies on the completion of a SELECT trigger that the query resulted in an EOF (End of Fetch) condition if the rows returned from the query do not equal the rows defined in the View Size. By default, the Database Browser task does not report an End of Fetch condition for a SELECT until a move operation advances the current row past the last row of the query.
-S# or -s#	Set maximum number (# = 4 to 160) of open SQL statements that the Database Browser will have active at one time. The default is 160. For very large applications, this program switch may have to be adjusted if the database server is unable to allocate a resource to open a new SQL cursor.
-V# or -v#	Set verbose level. (# = 0 to 1) Writes the SQL statements generated by the Database Browser to the log file. The Database Browser must have logging enabled for this program switch to work. The default is to not write the SQL statements to the log file.
-W# or -w#	Historian time-out parameter. (# = 5 to 30 seconds). Sets the maximum timeout in seconds for the Browser to wait for a response from the historian. The default is 30 seconds. For values less than 30 seconds, this switch will only work correctly when the historian initially achieves a successful connection with the database server. If the historian fails to successfully connect with the database server, Database Browser will time out in 30 seconds regardless of this switch setting.

STATUS CODES AND ERROR MESSAGES

The following table lists status codes written to a numeric Completion Status tag defined in the Database Browser Control table. The descriptive message is written to a message Completion Status tag and is also be written to the Task Message tag in the System Configuration table. See Chapter 13, “Historians” for status codes that are less than 100. The historian generates these codes, and they are returned to the Database Browser task when an historian operation is executed.

Code	Error	Cause	Action
100	Asynchronous error from Historian function.	An SQL COMMIT operation failed within the historian.	Consult the database administrator for the external database in use.
101	Error from Historian function.	A syntax error may have been made or information may not have been entered in a required field in a configuration table.	Correct the SQL statement syntax error by modifying the Information table for the task receiving the error. If the information is correct, ensure the database table exists.
102	No fields for select.	The task is trying to execute a select operation, but no tags have been defined to hold the data from the select operation.	Define some tags in the Tag Name field in the Database Browser Information table.
103	No fields for insert.	The task is trying to execute an insert operation, but no tags have been defined to hold the data from the insert operation.	Define some tags in the Tag Name field in the Database Browser Information table.
104	No fields for update.	Task is trying to execute an update operation, but no tags have been defined to hold the data from the update operation.	Define some tags in the Tag Name field in the Database Browser Information table.
105	Update and delete operations not supported with multi-table view.	Update and delete operations cannot be performed when using multi-table view.	Do not perform update and delete operations when using multi-table view.
106	Cannot update until select is performed.	A select trigger is defined but a select operation has not executed. Execute a select operation before an update operation.	Execute a select operation and then retry the update operation.

Code	Error	Cause	Action
107	Cannot delete until select is performed.	A select trigger is defined but a select operation has not been executed. Execute a select operation before a delete operation.	Execute a select operation and then retry the delete operation.
108	Cannot move until select is performed.	A select trigger is defined but a select operation has not been executed. Execute a select operation before a move operation.	Execute a select operation and then retry the move operation.
109	This row of data has been deleted.	A delete operation attempted on a nonexistant row	No action required.
110	A FactoryLink function returned an error.	FactoryLink PAK function encountered an unknown or unexpected error.	Contact your technical support representative.
111	A file function error occurred.	System encountered an unknown error while trying to read or write to the external buffer.	If not enough space, decrease the buffer size.
112	Bad tag in logical expression.	Either a typographical error exists or an undefined or invalid tag name is entered as an embedded variable tag in a Logical Expression field.	Correct typographical errors. If you did not make a typographical error, then define the tag in a FactoryLink task other than Browser.
113	Invalid use of tag in logical expression.	Logical expression does not contain a valid tag name.	Correct typographical errors and ensure the tag name is the name of a valid tag.
114	HSDA structure too small.	Invalid use of a tag in logical expression.	Define the tag used in the logical expression in a FactoryLink task other than Browser.
115	Can’t open log file.	Disk space too low.	If disk space is low, delete unneeded files or programs.
116	A request for memory failed.	Internal error.	Contact your technical support representative.
117	Can’t find unique index for table.	A positional update or delete operation occurred on a table without a unique index.	Create a unique index for the table and retry the operation.

One of the following messages is displayed to the right of BROWSER on the Run-Time Manager mimic if an error occurs with the Database Browser task or historian at run time. The first three letters (nnn) in the message below indicates whether the message came from the Database Browser task (DBB) or the Historian (HIS). See the .LOG file to display the complete message if it is truncated on the Run-Time Manager mimic.

Error Message	Cause and Action
nnn-[BAD_SMBX] Bad send mailbox. Browse rec: browse window name	Cause: You entered the wrong mailbox tag name. Action: Use the same tag name for the Historian Mailbox fields of the Historian Mailbox Information table and the Database Browser Control table.
nnn-[BAD_WHERE_TAG] Bad tag *tag name* for logical expression. Browse rec: browse window name	Cause: Either you made a typographical error or entered an undefined or invalid tag name as the embedded message tag in a logical expression. Action: Correct all typographical errors. Define the tag in another task if needed.
nnn-[CT_HDR] No sel, upd, or delete trigger defined. Browse rec: browse window name	Cause: Th select, delete, or update trigger is not defined. Action: Define at least one of the triggers listed above.
nnn-[DBTBL_SYNTAX] The Database Table value is missing a ".". Browse rec: browse window name	Cause: You left the "." out of the entry in the Database.Table Name field in the Database Browser Control table. Action: Put a "." between the database name and the table name in the entry in the Database.Table Name field.
nnn-[FILE_ERR] File function "function" returned error *error code*. Browse rec: browse window name	Cause: The system encountered an unknown error while trying to read from or write to the external buffer. Action: Verify you have enough disk space. If not, decrease the buffer size.
nnn-[FL_FUNC] Function "function" returned error *error code*. Browse rec: browse window name	Cause: The FactoryLink PAK function encountered an unknown or unexpected error. Action: Report the error to your support representative.
nnn-[FL_FUNC] Function "FL_WRITE" returned error 9. Browse rec: browse window name	Cause: The column type in the database FactoryLink is trying to read from does not match the column type defined in the Database Logging task Schema Creation table. Action: Make the column type in the Column Type field of the Schema Creation table the same as in the database.

Error Message	Cause and Action
nnn-[FLERROR] FactoryLink error *error number*	Cause: An unknown PAK function error occurred within the historian. Action: Report the error to your support representative.
nnn-[HSCONNECT] Failed to connect to historian	Cause: You may have specified the wrong mailbox tag name. Action: Specify the Historian Predefined mailbox tag name in the Historian Mailbox field in the Database Browser Control table.
nnn-[HSDA_TOO_SMALL] HSDA structure too small. Browse rec: browse window name	Cause: You specified an invalid tag in a logical expression. Action: Make sure that all tags used in the logical expression are defined.
nnn-[HSDBERROR] Historian database error: error message	Cause: This message is accompanied by various other messages that describe the cause of the error. Action: Read the accompanying message displayed on the Run-Time Manager screen or in the .LOG file and attempt to correct the problem based on the instructions in the message.
nnn-[HSDUPLICATE] Tried to insert a duplicate row	Cause: You tried to insert a duplicate row into a result table. Action: No action required
nnn-[HSENDOFETCH] Last row fetched or row not found	Cause: The task could not find a row during an update operation because that row does not exist. Or, during a move or position operation, you specified a nonexistent row (for example, row 100 when the table has only 50 rows). You attempted to go past the end or above the beginning of the result table. Action: No action required
nnn-[HSFLDEXISTS] Tried to add an existing field	Cause: You tried to add an existing field. Action: No action required
nnn-[HSMAXOPENS] Too many open sessions	Cause: You tried to open more than 10 unique databases. Action: Use fewer than 10 databases in a configuration table.
nnn-[HSMEMORY] Memory error malloc failed	Cause: Not enough memory is allocated for the historian. Action: Allocate more memory for the historian.

Error Message	Cause and Action
nnn-[HSNOFIELD] Tried to access a nonexistent field	Cause: You tried to open a nonexistent field. Action: No action required
nnn-[HSNOTABLE] Tried to access a nonexistent table	Cause: You tried to open a nonexistent table. Action: No action required
nnn-[HSPREPARE] Failed to prepare stmtid	Cause: A nonexistent table name or field name is specified or a syntax error is made in an SQL statement. Action: Ensure all entries in the Database Browser table are correct, especially those for the SQL statement.
nnn-[HSSTMTID] Invalid stmtid returned from historian	Cause: The historian shut down before the Database Browser task or another historian-client task. Action: Shut down the Database Browser, then all other historian-client tasks, and then the historian. Restart the historian, followed by the Database Browser and other historian-client tasks.
nnn-[HSTBLEXISTS] Tried to create an existing table	Cause: You tried to create an existing table. Action: No action required
nnn-[HSTIMEDOUT] Historian task not responding. Maximum timeout exceeded	Cause: A browser request did not get a response from the historian within the timeout period. Action: If the timeout period is less than the time taken to serve the request, you may increase the timeout, e.g., from -w300 to -w400.
nnn-[HSUNKNOWN] Unknown function request sent to historian	Cause: An error occurred within Browser. Action: Report the error to your support representative.
nnn-[INVUSE_WHERE_TAG] Invalid use of tag in logical expression. Browse rec: browse window name	Cause: A logical expression contains an invalid tag name. Action: Correct all typographical errors and ensure the tag name is the name of a valid tag.
nnn-[MULTI-VIEW] Update and delete operations not supported with multi-table view. Browse rec: browse window name	Cause: You tried to perform an update or delete operation while using multi-table view. Action: Do not try to perform update or delete operations

Error Message	Cause and Action
nnn-[NO_FLDS_INS] No fields for insert. Browse rec: browse window name	Cause: The task is trying to perform an insert operation but no tags are defined to hold the data from the insert operation. Action: Define some tags in the Tag Name field of the Database Browser Information table.
nnn-[NO_FLDS_SEL] No fields for select. Browse rec: browse table name	Cause: The task is trying to execute a select operation but no tags are defined to hold the data from the select operation. Action: Define some tags in the Tag Name field of the Database Browser Information table.
nnn-[NO_FLDS_UPD] No fields for update. Browse rec: browse table name	Cause: The task is trying to execute an update operation, but no tags have been defined to hold the data from the update operation. Action: Define some tags in the Tag Name field of the Database Browser Information table.
nnn-[NO_LOGICAL_EXPR] No logical expr. Browse rec: browse table name. Column name: column name	Cause: A logical operation was defined but the logical expression was not specified. Action: Either delete the operation or create a logical expression.
nnn-[NO_MEMORY] Out of RAM	Cause: Not enough RAM is available to run this task. Action: Allocate more RAM for the Database Browser task.
nnn-[NOTASSOC] Col name column name not associated with browse or where tag. Browse rec: browse window name	Cause: A non-existent or an invalid tag is specified. A column name is also specified but not a logical expression. Action: Define a tag in the Tag Name field of the Database Browser Information table and/or specify a logical expression.
nnn-[NO_UNIQ_INDX] Can't find unique index for table. Browse rec: browse window name	Cause: You attempted a positional update or delete operation on a table without a unique index. Action: Create a unique index for the table and retry the operation.
nnn-[NULL_ROW] This row of data was deleted. Browse rec: browse window name	Cause: You attempted a delete operation on a deleted row. Action: No action required

Error Message	Cause and Action
nnn-[NULL_TABLE] No data for this table. Browse rec: browse table name	Cause: You tried to perform an update, move, or delete operation on a result table that contains no rows of data because the select operation resulted in no rows of data or you deleted all rows of data from the table. Action: No action required
nnn-[OPEN_LOG] Can't open .LOG file.	Cause: The computer may have run out of disk space. Action: Delete any unnecessary files or programs.
nnn-[SEL_B4_DEL] Can't delete until select is performed. Browse rec: browse window name	Cause: You defined a select trigger but a select operation was not executed. A select operation must be executed before a delete operation can be performed. Action: Execute a select operation before the delete operation.
nnn-[SEL_B4_MOVE] Can't move until select is performed. Browse rec: browse window name	Cause: A select trigger is defined but a select operation was not executed. A select operation must be executed before a move operation can be performed. Action: Execute a select operation before the move operation.
nnn-[SEL_B4_UPD] Can't update until select is performed. Browse rec: browse window name	Cause: A select trigger is defined but a select operation was not executed. A select operation must be executed before an update operation can be performed. Action: Execute a select operation before the update operation.
nnn-[SQL_ASYNC] Asynchronous failure to name. Error: error message.	Cause: An SQL COMMIT operation failed in the historian. For Oracle, it can fail if you do not have enough disk space. Action: Consult the administrator for the database in use.
nnn-[SQL_SYNC] Historian function function failed. Error: error message Browse rec: browse window name	Cause: A syntax error was made or information for a required field was not specified in a configuration table. Action: Modify the information in the Database Browser Information table to create a correct SQL statement if the error is a syntax error. Ensure the database table exists.
nnn-[UNSOL_MSG_RCVD] Unsolicited message received from tag number	Cause: A task wrote to Browser's mailbox tag unexpectedly. Action: Check the X-reference list to see which task is using the same tag and change the mailbox tag name of the task.

Error Message	Cause and Action
F_BAD_CT_SIZE Bad size for CT #(CT number)	Cause: There is a discrepancy between the Database Browser script file and the size. You may have modified the Browser .CTG file. Action: Copy the .CTG file from the Installation disk over the modified one.
F_BAD_RMBX Invalid global mailbox: mailbox name	Cause: Browser's predefined mailbox tag is nonexistent in the GLOBAL.CT file. Action: Report the error to your support representative.
F_INIT Task initialization failed	Cause: This message is preceded by another error message that explains the cause of the error. Action: Examine the preceding message to determine the cause of the initialization failure.
F_NO_CTS No valid tables in CT archive file name	Cause: The Database Browser tables have not been configured. Action: Configure the Database Browser tables.
F_NO-DOMAIN_NAME No domain name for appl. directory *directory*	Cause: No domain name is specified for the application directory. Action: Specify a domain name for the application directory.
F_OPEN_CT Can't open CT archive file name	Cause: A .CT file may have been deleted. Cause: Use CTGEN to rebuild the .CT file.
F_READ_CT Can't read CT #CT number in CT archive file name	Cause: A .CT file is corrupt. Action: Rebuild the corrupt .CT file.

Chapter 6 . . . .

Database Logger

The Database Logger task (Logger) writes blocks of data to a historical database to preserve data for historical purposes. Each time a new value for a tag is collected or computed, the current value of the tag in the real-time database is overwritten by the new data. To preserve this data, the Database Logger task reads the data from the real-time database and sends it to a disk-based relational database through a historian.

The historian used for this transfer depends on the relational database receiving the data. The database can be either the SQL Server database that can be purchased with FactoryLink or a third-party relational database, such as Oracle.

With the Logger, you can create a table and specify which tags to capture in that table. When the value of any tag changes, the values of all tags in the table are logged. Database Logging provides the ability to group tags in a database table, and event-based data can be logged using a sequence key rather than a time key.

Database Logging is best for situations when you want to:

- Log all tags when any tag changes or is triggered
- Simultaneously log a group of like (logically associated) tags
- Group data logged based on some criteria
- Configure a table column to be a dynamic pen on a trend chart

Data is logged using logging operations. Each logging operation defines which data to log when the operation executes. The operations and the data for logging with each operation are defined in tables:

- Database Logging Control table—Defines the operations that log data
- Database Logging Information table—Defines which data to log with each operation

Database Logging Methodology

This section describes and illustrates how real-time tags are logged to a relational database.

1. The real-time database receives and stores data in tags from various sources, such as a remote device, user input, or computation results from a FactoryLink task. When data is collected and stored in this database, other tasks can access and manipulate it.

2. The Logger reads the values of tags in the real-time database and maps the tags to columns in a disk-based relational database table.

3. The Logger sends the data from the real-time database to an historian mailbox in the form of an SQL `INSERT` statement. The request remains in the historian mailbox until historian processes the request.

4. After historian processes the request, it connects to the relational database and inserts the data in the relational database file, where other applications can use the data.

You must consider the following when selecting a logging method:

Grouping Data: FactoryLink permits you to log data with or without a group association. If you elect to log data without a group association (nongrouped), the table delineates the data in the table from data in other tables.

Ordering Data FactoryLink permits you to log grouped and nongrouped data with or without a sequence or order number. If you elect to associate a sequence number with rows of data, your table structure must include a column that contains the sequence number associated with the row of data (record). The sequence number can be an integer or a time-stamp.

Indexing Records are displayed in a relational database table in the order they are added. This is known as the natural or physical order. The field or fields selected for indexing are called the index key.When a table is indexed, an index file is created that contains the indexed order. The actual record order in the database file remains unchanged. Accessing indexed records is faster than accessing non indexed records.

Deletion Methods The logging method you choose depends in part on how data gets deleted from the relational database. Your options include:

Record rollover

Subgroup rollover

Group data deletion

Note: If you're using the dBase IV historian, record rollover controls table size. If you're using another database, you should use a stored procedure and PowerSQL to control table size.

Database Logging Methods Available database logging methods include:

Nongrouped/Nonsequenced Method

Nongrouped/Sequenced Method

Grouped Method

This chapter describes how to configure logging and provides sample applications for the following grouping methods:

Nongrouped/Nonsequenced Data

Use this logging method if you want to log data without a group association and logging order is unimportant.

The sample application for nongrouped/nonsequenced data is for a gasoline station that logs the following data for an unleaded fuel storage tank:

- Tank level
- Tank pressure
- Tank temperature

Nongrouped/Sequenced Data

Use this logging method if you want to log data without a group association and want to know the logging sequence.

The sample logging operation for nongrouped/sequenced data is for a gasoline station that logs the total gallons of unleaded and diesel gas pumped each hour of the day for 24 hours. The hourly total is an accumulated value stored in a real-time database tag. Each time gas is pumped, the number of gallons sold is added to the accumulated value.

The total for each type of gas (unleaded and diesel) is logged to different columns in the same table.

Grouped and/or Subgroup Data

Use this logging method if you want to log data with a group name, group name and subgroup number, or subgroup number.

The sample logging operation is for a gasoline station that logs the total gallons of unleaded and diesel gas sold each hour of the day for each day of the week for a week. The hourly total is an accumulated value stored in a real-time database tag. Each time gas is pumped, the number of gallons sold is added to the accumulated value.

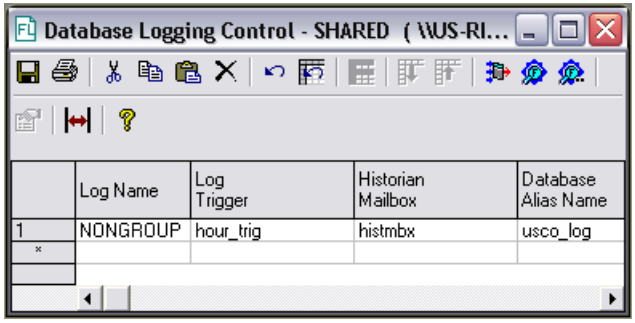
The total for each type of gas (unleaded and diesel) is logged to the same column in the same table but distinguished by a *groupname_subgroupnum* in the group column.

Each day of the week is represented by the *subgroupnum* that increments at the end of each day. The table size is controlled by subgroup rollover that occurs after seven days.

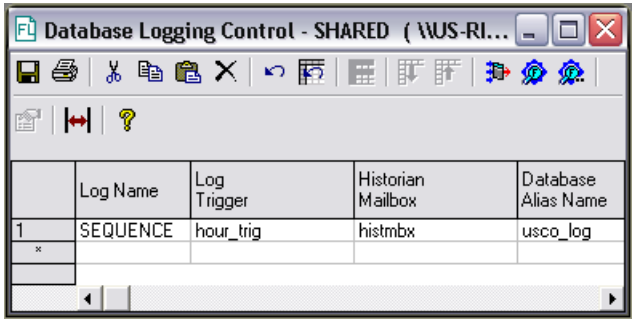
DATABASE LOGGING CONTROL TABLE

The Database Logging Control table sets up the mailbox the Logger uses to communicate to the historian as well as the database alias and table information that Logger needs to pass to historian.

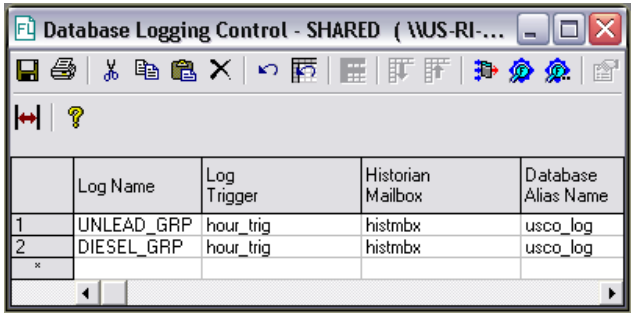
The following examples are used with the discussion of the field definitions.



Nongrouped/ Nonsequenced Data



Nongrouped/ Sequenced Data



Grouped and/or Subgrouped Data

Accessing

In your server application, open **Data Logging > Database Logging > Database Logging Control**.

Field Descriptions

Log Name — Name to reference the logging operation.

The nongrouped/nonsequenced data example defines one logging operation named NONGROUP that logs nongrouped tank data for the unleaded fuel storage tank.

The nongrouped/sequenced data example defines one logging operation named SEQUENCE that logs the total gallons of unleaded and diesel gas sold each hour.

The grouped and/or subgrouped data example defines two logging operations: one named UNLEAD_GRP that logs total gas pumped each hour for unleaded gas and DIESEL_GRP that logs total gas pumped each hour for diesel gas.

Valid Entry: alphanumeric name of up to 16 characters

Log Trigger — Tag name that triggers the logging operation. If you leave this field blank, the logging operation activates with the **Log-On Change** field in the Database Logging Information table.

In the nongrouped/nonsequenced data example, the NONGROUP operation executes when the **hour_trig** tag is set.

In the nongrouped/sequenced data example, the SEQUENCE operation executes when the **hour_trig** tag is set.

In the grouped and/or subgroup data example, both operations execute when the **hour_trig** tag is set.

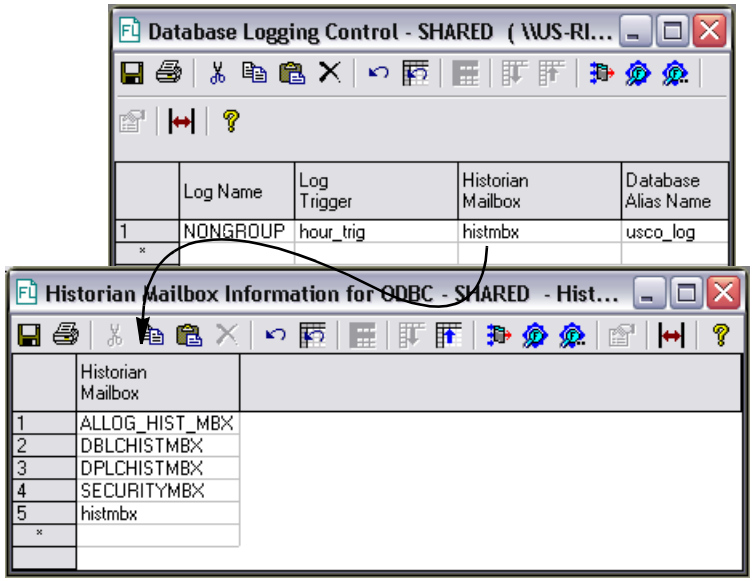
Specify both a field trigger and a log-on change in the **Log On Change** field of the Database Logging Information table to enable the same operation. The operation executes each time the trigger is set and each time the tag value changes.

Valid Entry: tag name

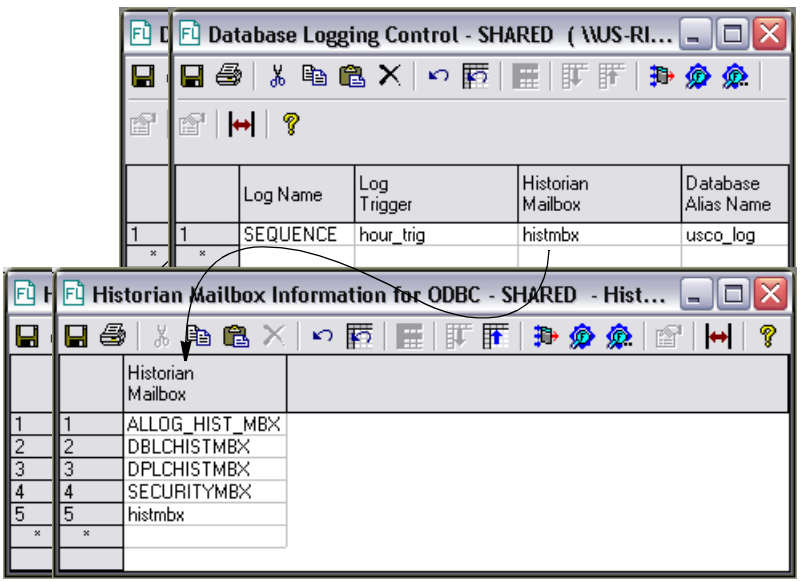
Valid Data Type: digital, analog, longana, float, message

Historian Mailbox — Mailbox tag name the historian uses to transfer data. You must define the connection for this mailbox in the historian table.

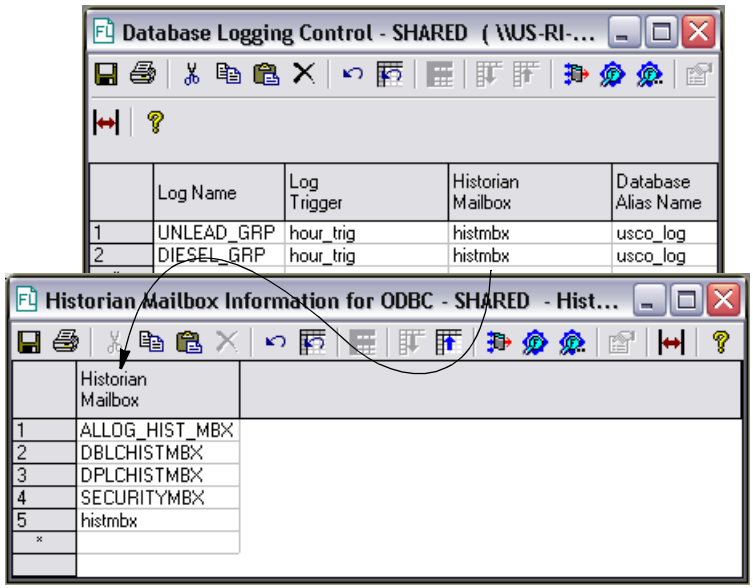
Nongrouped/ Nonsequenced Data



Nongrouped/ Sequenced Data



Grouped and/or Subgrouped Data



For more information on the strategy for defining historian mailboxes, see “Historians” on page 253.

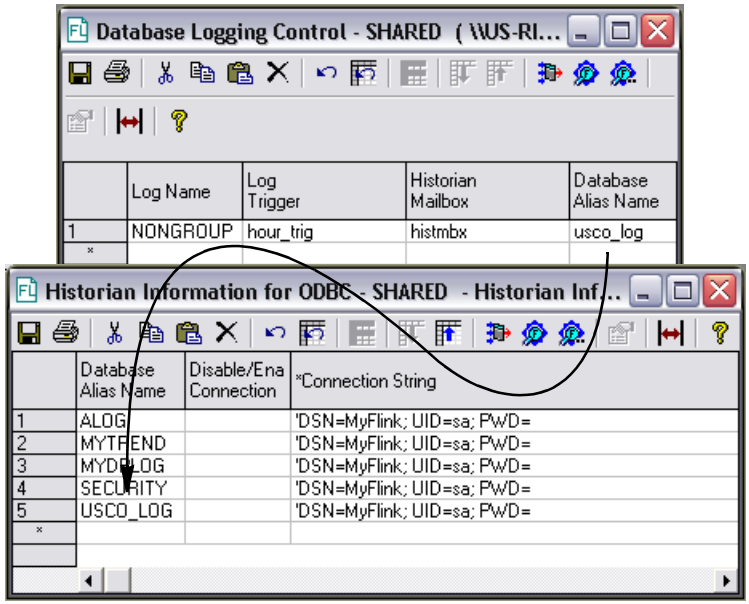
Valid Entry: mailbox tag name

Database Alias Name

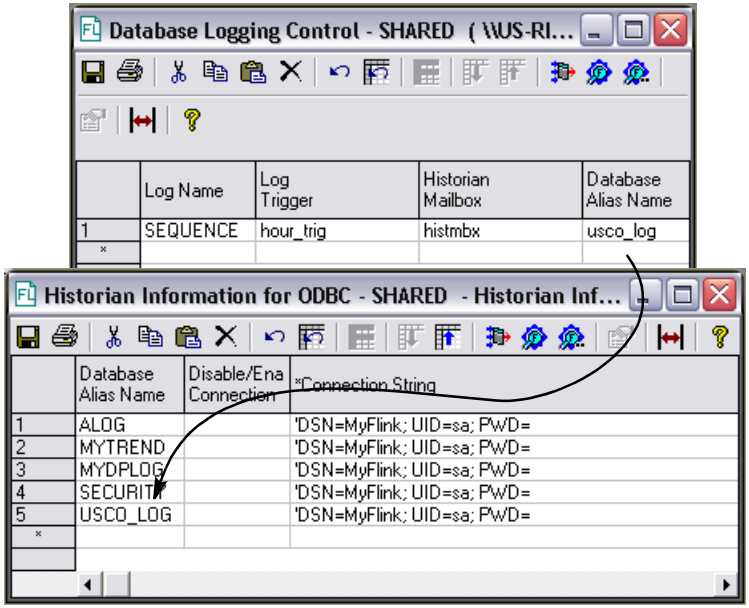
Database alias name that references the database where historian sends the data. You must define the same alias name in the historian table when defining the mailbox connection. In the examples, the data is logged to the USCO_LOG database, which is an alias for referencing the USCO database.

Valid Entry: database alias name

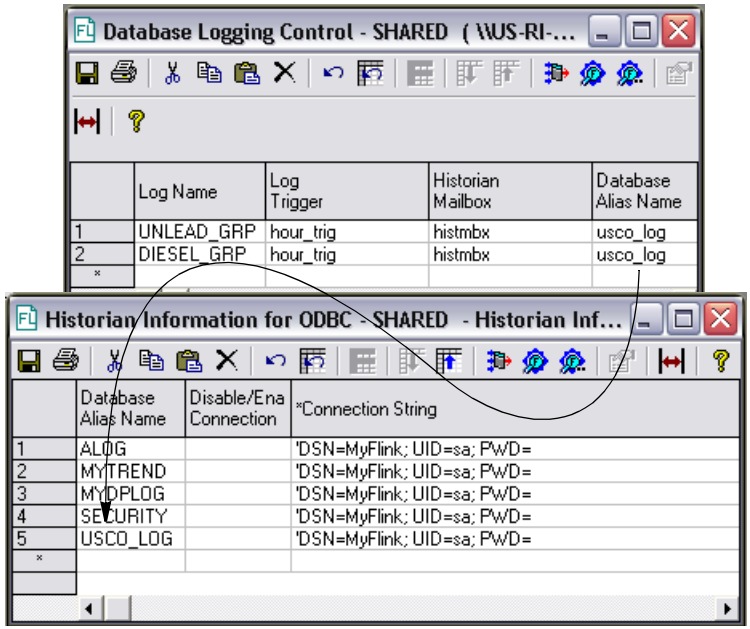
Nongrouped/ Nonsequenced Data



Nongrouped/ Sequenced Data



Grouped/ Subgrouped Data



Database Table Name

Unique name to reference the table in the relational database that receives the data. If this table does not already exist in the relational database when data is logged to it, it is created using the schema defined in the **Schema Name** field.

Valid Entry: unique name of up to 31 characters

Schema Name

Name that defines the relational database table structure that receives the data. If the table specified in the **Database Table Name** field does not already exist in the relational database when data is logged to it, it is created using the schema defined in this field.

If you enter a name in this field, it must match the name defined in the **Schema Name** field on the Schema Control table. If it does not match a defined schema name, it is as if you left this field blank.

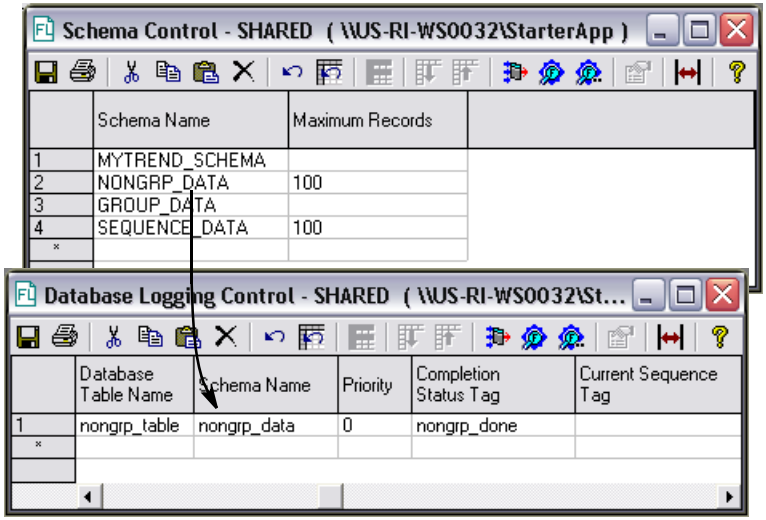
A table is not created and data is not logged if you leave this field blank.

Valid Entry: schema name

Nongrouped/ Nonsequenced Data

In this example, the **nongrp_data** schema is used.

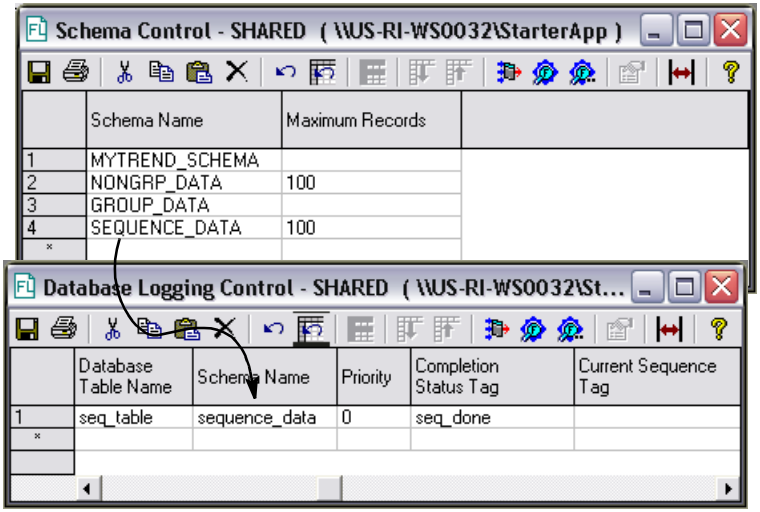
Data is logged to the database table named **nongrp_table**.



Nongrouped/ Sequenced Data

In this example, the **sequence_data** schema is used.

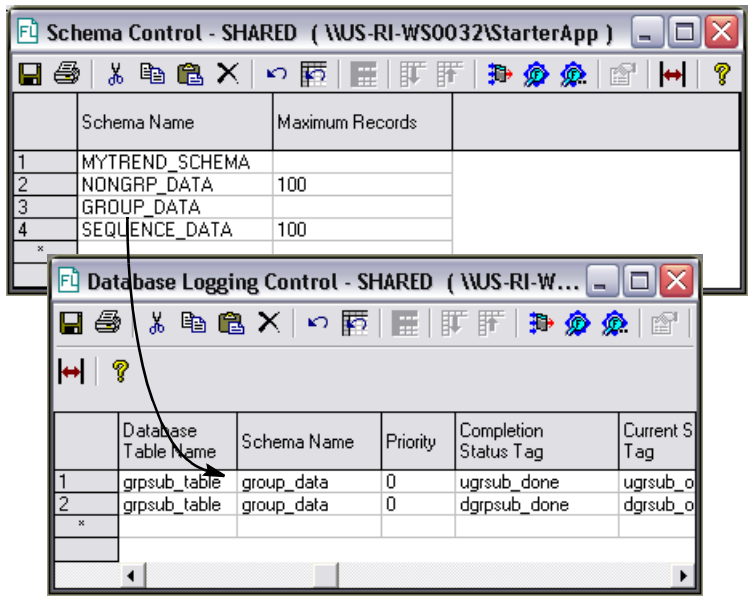
Data is logged to the database table named **seq_table**.



Grouped/ Subgrouped Data

In this example, the **group_data** schema is used.

Data is logged to the database table named **grpsub_table**.



Priority — Number that controls the order in which Logger handles the queueing of logging operations. This is a relative priority to other logging operations. For example, if Logger receives two requests at the same time from two different operations, it processes the request with the highest priority (lowest number) first.

Valid Entry: 0 to 9
Default: 0

Disable Tag — Tag that disables logging operations. This must be in the Shared domain.

Valid Entry: tag name
Valid Data Type: digital, analog, float, or longana
Default Entry: digital

Completion Status Tag — Tag updated by Logger to indicate the completion of this logging operation. A 1 is written to this tag after data is logged to the historian mailbox. Do not specify a tag name if you do not want to track the completion of this logging operation.

This field is typically used for coordinating activities between tasks. That is, you can use the completion of this logging operation to trigger another task or operation.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float

In the nongrouped/nonsequenced data example, the remaining fields are not needed and are left blank.

In the nongrouped/sequenced data or group and/or sequence data, the following two fields are only used if you are using an integer sequence number to order the logged data.

Current Sequence Tag

Tag name that stores the current sequence number for this logging operation. The next time a logging operation occurs, the value stored in this tag is incremented by one and the new value is assigned to the row of data.

The **Current Sequence Tag** value is saved in a file when FactoryLink is not running and updated to this field when FactoryLink is restarted, so interrupts do not cause problems with duplicate records in the relational database table.

Valid Entry: tag name
Valid Data Type: analog, longana, float

Sequence Change Tag

Tag name used to adjust the numbering sequence during run time when ordering data by integer. This field provides the ability to manually create gaps in the numbering sequence of ordered data.

If you leave this field blank, you can never manually adjust the numbering sequence of ordered data.

When a value is force written to the tag specified in this field, the value written is used to adjust the **Current Sequence Tag** value. For example, if the **Current Sequence Tag** value is 5 and you force write a value of 5 to this tag, the **Current Sequence Tag** changes to 10.

If the tag type is digital, increment the **Current Sequence Tag** by 1 by force writing a 1 to the **Sequence Change Tag**.

If the tag type is analog, long analog, or float, force write any whole positive or negative number to the **Sequence Change Tag**. If you write a positive number, the value of the **Current Sequence Tag** increments by the specified number. If you write a negative number, the value of the **Current Sequence Tag** decrements by the specified number.

Be careful when adjusting the **Current Sequence Tag** by a negative number as this can cause two rows of data to have the same sequence number. If the sequence number is part of a unique index, duplicate record values are not permitted in the column. If an attempt is made to log a duplicate value, FactoryLink displays an error message and the data is not logged.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float

In the nongrouped/sequenced data example, the remaining fields are blank. They are not needed for nongrouped/sequenced data.

In the grouped and/or subgrouped data example, the following fields are only needed if you are using an integer sequence number to order the logged data.

Group Delete Tag

Tag name that contains the group name to delete when the **Group Delete Trigger** tag executes. When group deletion is triggered, all group data is deleted. If the group contains subgroups, all subgroups and their data are deleted.

If you do not specify a tag in this field, you can never delete data by specifying a *groupname*. If you do not specify a tag in this field but do specify a **Group Delete Trigger**, all nongrouped data sequenced by integer and all subgroups without a *groupname* association are deleted from the table when the trigger executes.

Enter the name of the group to delete in the **Value** field of the Tag Editor.

Valid Entry: tag name
Valid Data Type: message, digital, analog, longana, float

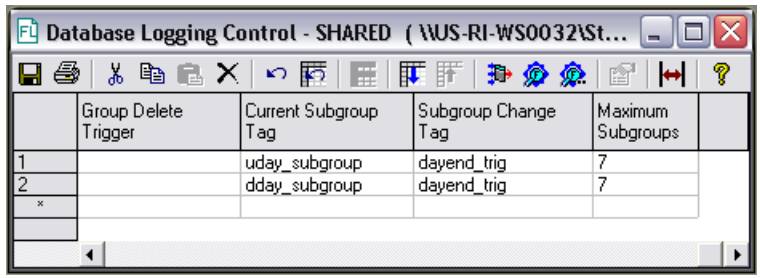
Group Delete Trigger

Tag name that triggers the deletion if you want to delete data by group based on an event. When this trigger is set, the data for the group specified in the **Group Delete Tag** field on this table is deleted.

If you leave this field blank, deletion of group data is never triggered.

Leave the **Group Delete Tag** and **Group Delete Trigger** fields blank if you are subgrouping data without a group association. If you wish to delete subgroup data without a group association, use the subgroup rollover feature to delete nongrouped data sequenced by integer.

The following table shows the fields in the Database Logging Control table displayed after the **Group Delete Trigger** field. Leave the fields blank if you are not using subgrouping.



Valid Entry: tag name

Valid Data Type: digital, analog, longana, float

Note: Misleading Error When Doing GROUP DELETE:

A SERIAL table is not created if Logger is configured to log GROUP data without using a sequence column. However, when the trigger for GROUP DELETE is activated, an attempt is made to delete the appropriate entry from the SERIAL table for this group. This attempt produces an error, because it tried to access a nonexistent table. The data is correctly deleted from the database table, and the error has no effect on the historian and/or Logger operations that follow.

Current Subgroup Tag

Tag that stores the current subgroup number for this group.

The next time a new subgroup is created for this group, the value stored in this tag is adjusted by the amount defined in the **Subgroup Change Tag** field on this table and assigned to the new subgroup.

If this field is left blank, subgroups are not supported for this group and no subgroups are created for this group.

The value of this tag is set to 0 at group delete and 1 at subgroup rollover.

The **Current Subgroup Tag** value is saved in a file when FactoryLink is not running and updated to this field when FactoryLink is restarted so interrupts do not cause problems with duplicate records in the relational database.

Valid Entry: tag name

Valid Data Type: analog, longana, float

Subgroup Change Tag

Tag that triggers a subgroup change. When the tag value changes, a new subgroup is created. The value to assign to the new subgroup is calculated by adjusting the value in the **Current Subgroup Tag** by the value in the **Subgroup Change Tag**. This field provides the ability to manually set or change the subgroup numbering sequence.

If you enter a tag name in this field, a tag must also be defined in the **Current Subgroup Tag** field. Leave this field blank if you are not using subgrouping.

If the tag type is digital, the **Current Subgroup Tag** increments by 1 when 1 is force written to this tag.

If the tag type is analog, long analog, or float, force write any whole positive or negative number to this tag. If it contains a positive number, the value in the **Current Subgroup Tag** increments by the specified number. If it contains a negative number, the value of the **Current Subgroup Tag** decrements by the specified number.

Be careful when adjusting the **Current Subgroup Tag** by a negative number. This can cause two subgroups to have the same number. If the group ID is part of a unique index, duplicate record values are not permitted in the column. If an attempt is made to log a duplicate value, FactoryLink displays an error message and the data is not logged.

Leave this field blank to use the same subgroup number each time this trigger executes.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float

Maximum Subgroups

Maximum number of subgroups allowed before subgroup rollover occurs. If you leave this field blank but have specified a subgroup tag, the Logger continues to increment the subgroup number indefinitely until the disk is full.

You must define a tag in the **Current Subgroup Tag** field for subgroup rollover to occur.

Valid Entry: numeric value

DATABASE LOGGING INFORMATION TABLE

The data to log with each operation is defined in the Database Logging Information table. This table also maps the data to specific columns in the relational database table receiving the data. This section describes how to complete the Database Logging Information table to log data for the sample application to one of the following tables.

Nongrouped/Nonsequenced Data

gallons	pressure	temp

Nongrouped/Sequenced Data

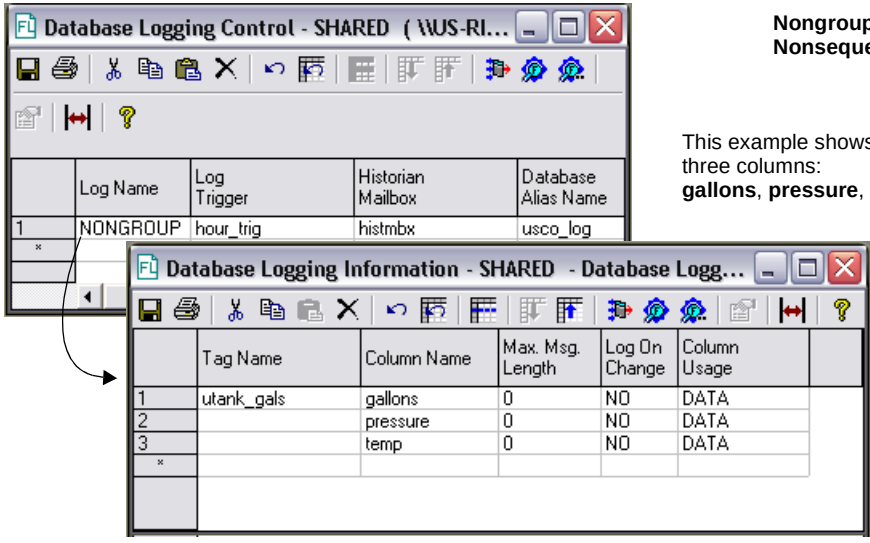
Order_col	Time_col	Ugal_sold	Dgal_sold

Grouped and/or Subgrouped Data

group_ID	order_col	gals_sold

Accessing

In your server application, open **Data Logging > Database Logging > Database Logging Control > "your trend" > Database Logging Information**.

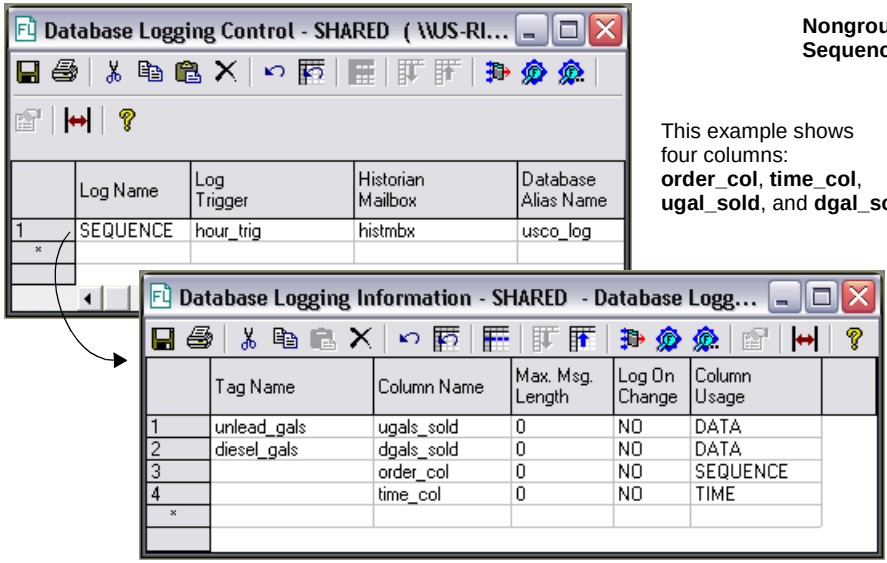


Nongrouped/ Nonsequenced Data

This example shows three columns: **gallons**, **pressure**, and **temp**.

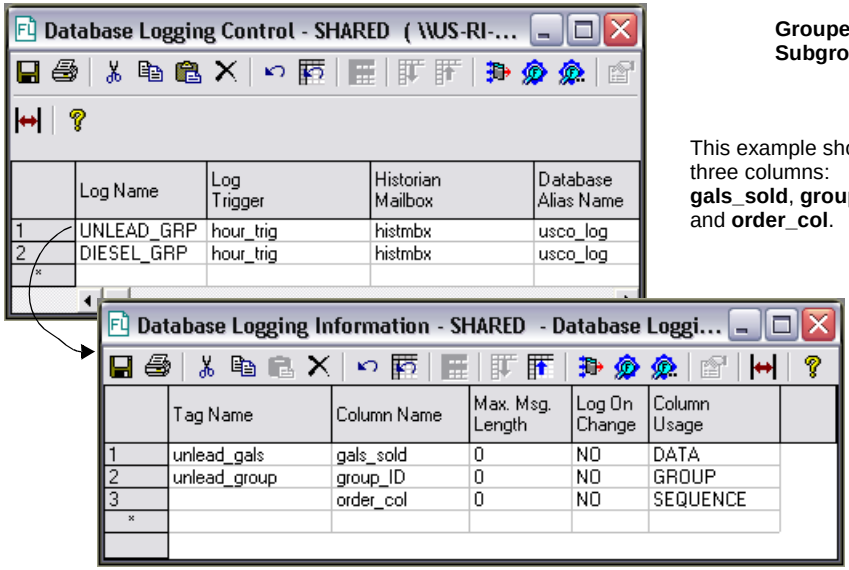
Nongrouped/ Sequenced Data

This example shows four columns: **order_col**, **time_col**, **ugal_sold**, and **dgal_sold**.



Grouped/ Subgrouped Data

This example shows three columns: **gals_sold**, **group_ID**, and **order_col**.



Field Descriptions

Specify the following information for this table:

In the nongrouped/nonsequenced data:

Tag Name — Tag name that references the tag to log in the column this entry represents.

Valid Entry: tag name

Valid Data Type: digital, analog, longana, float, message

In the nongrouped/sequenced data:

Tag Name — What you enter in this field depends on the usage of the column that receives the logged data. The options are described in the following table.

Column Use	Tag Name Field Value
data	Tag to log in the column represented by this entry
sequence	If you want to manually set the sequence number to use for the next logging operation, specify the tag that contains the absolute number you want to use. If you never want to manually set the sequence number, leave this field blank. If this tag value is any number except 0, that number is assigned as the sequence number for the next row of data logged. Then this tag value is reset to 0. If a record with that sequence number already exists and the sequence number is part of a unique index, data is not logged. If this tag value is 0, the next sequence number assigned to a row of data logged is the value in the **Current Sequence Tag** field after it is incremented by one. This operation does not update the **Current Sequence Tag** field. The next sequence number assigned continues from the number in the **Current Sequence Tag** field.
time	Do not specify a tag name. SECTIME is always used.

The data type of the tag defined depends on the column usage. The options are described in the following table:

Column Usage	Valid Tag Types
data	digital, analog, longana, float, message
sequence	analog, longana, float
time	The global tag SECTIME is already defined. You do not need to specify a type.

In the grouped and/or subgrouped data:

Tag Name

What you enter in this field depends on the usage of the column that receives the logged data. The options are described in the following table.

Column Use	Tag Name Field Value
data	Tag to log in the column represented by this entry
group	Tag name that contains the name of the group associated with this operation.
time	Do not specify a tag name. SECTIME is used.
sequence	If you want to manually set the sequence number to use for the next logging operation, specify the tag name that contains the absolute number you want to use. If you never want to manually set the sequence number, leave this field blank. If this tag value is any number except 0, that number is assigned as the sequence number for the next row of data logged. Then this tag value is reset to 0. If a record with that sequence number already exists and the sequence number is part of a unique index, data is not logged. If this tag value is 0, the next sequence number assigned to a row of data logged is the value in the **Current Sequence Tag** field after it is incremented by one. This operation does not update the **Current Sequence Tag** field. The next sequence number assigned continues from the number in the **Current Sequence Tag** field.

The data type of the tag defined depends on the column usage. The options are described in the following table:

Column Use	Valid Tag Types
data	digital, analog, longana, float, message
sequence	analog, longana, or float
time	The global tag SECTIME is already defined. You do not need to specify a type.
group	If groupname is numeric, use digital, analog, longana, or float. If groupname is made up of characters, use message.

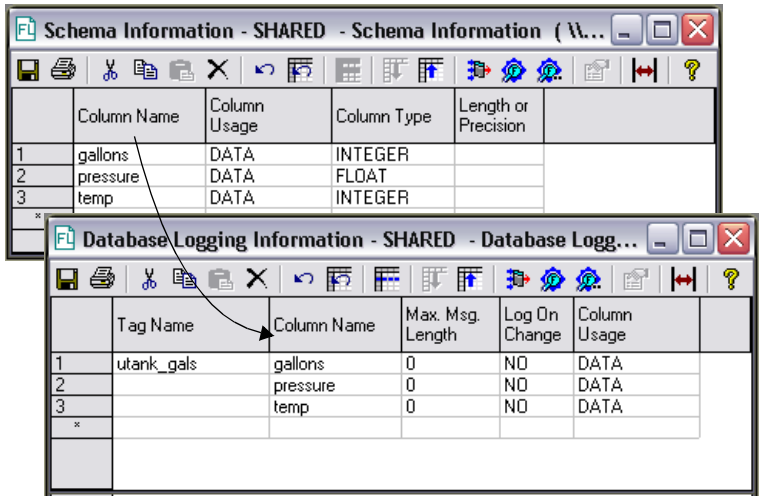
Column Name — Name of the column to receive the data specified in the **Tag Name** field.

If the table structure is defined through the relational database software, the name must match one of the columns defined for the table.

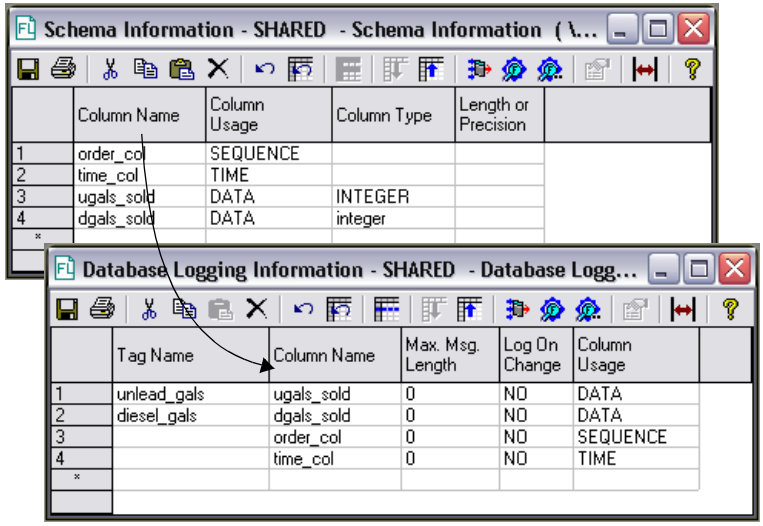
If the table structure is defined through FactoryLink, the name must match one of the column names defined in the **Column Name** field on the Schema Information table. This relationship is shown in the following graphics.

Valid Entry: column name

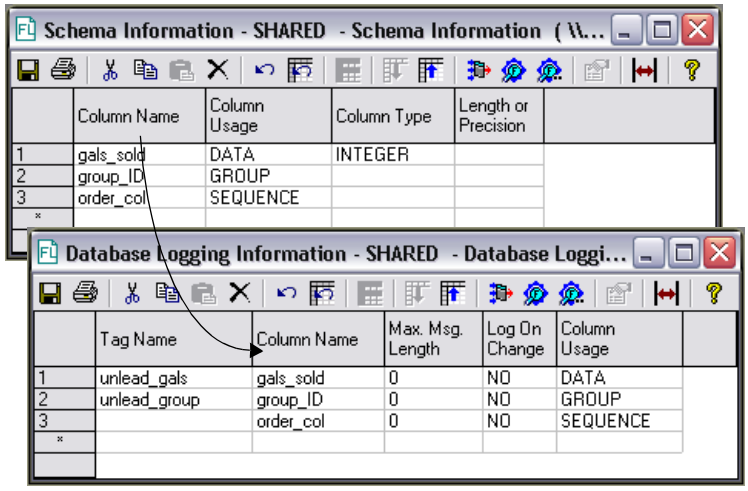
Nongrouped/ Nonsequenced Data



Nongrouped/ Sequenced Data



Grouped/ Subgrouped Data



Max. Msg. Length

Maximum number of characters allowed in the column. If you leave this field blank when the tag type is message, a default of 80 is assumed. The value specified in this field should always be equal to or less than the value specified in the **Length or Precision** field on the Schema Information table.

Leave this field blank to assume a default of 0 for tags defined with a tag type other than message, which indicates this field is not used.

Valid Entry: 1 to 999 (if message)
Default: 80

Log On Change

Specify whether or not the logging operation is triggered if the specified **Tag Name** field value changes. This can be one of the following:

YES When the tag value changes, the log operation occurs. If you leave the **Log Trigger** field blank on the Database Logging Control table, you must indicate YES in this field for at least one tag on this table to activate the logging operation.

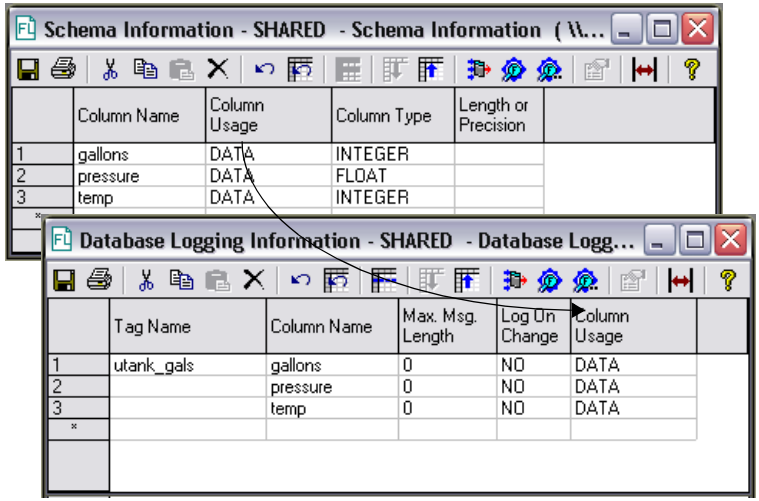
NO When the tag value changes, the log operation does not occur.

Specify both a trigger in the **Log Trigger** field of the Database Logging Control table and a login change in this field for the same operation. The operation executes each time the trigger is set and each time the value specified changes.

Column Usage — Kind of information sent to the column.

In the nongrouped/nonsequenced data example, column usage is data for each column in a table logging nongrouped/nonsequenced data.

Nongrouped/ Nonsequenced Data



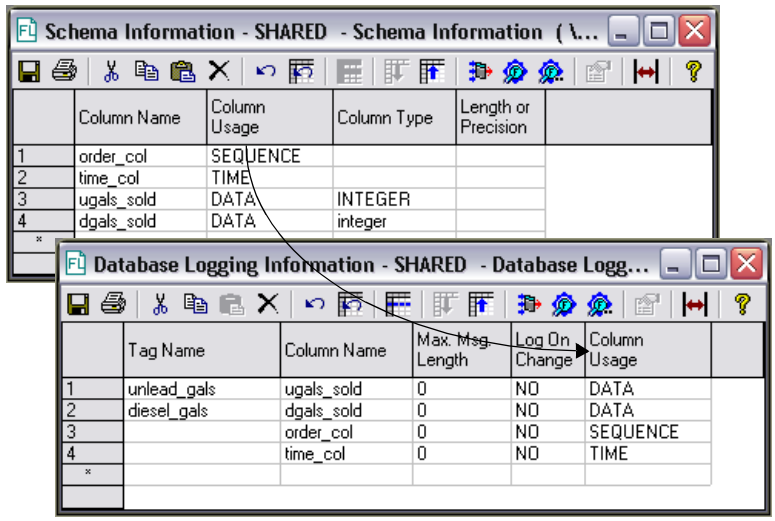
In the nongrouped/sequenced data example, column usage is one of the following:

data — Use for columns that do not receive a sequence number in the form of an integer or time-stamping.

sequence — Use for columns receiving a sequence number in the form of an integer.

time — Use for columns receiving a sequence number in the form of a time-stamping. The time-stamping used is SECTIME.

Nongrouped/ Sequenced Data



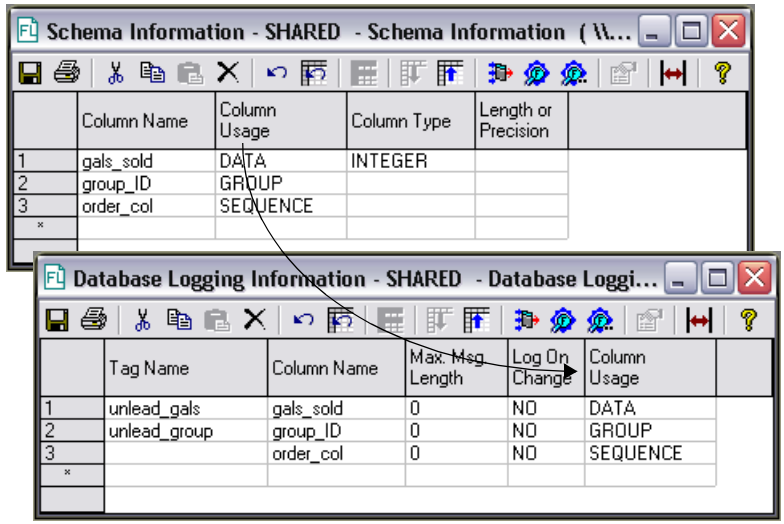
In the grouped and/or subgrouped data example, column usage is one of the following:

DATA	Use for columns that do not receive a sequence number or group ID.
SEQUENCE	Use for columns receiving a sequence number in the form of an integer.
TIME	Use for columns receiving a sequence number in the form of a time-stamping. The time-stamping used is SECTIME.
GROUP	Use for columns receiving the group identification assigned to the row of data.

If you leave this field blank, data is assumed even though the word data does not display in the field.

If the table structure is defined through FactoryLink, the value entered in this field must match the value entered for this column in the **Column Usage** field on the Schema Information table.

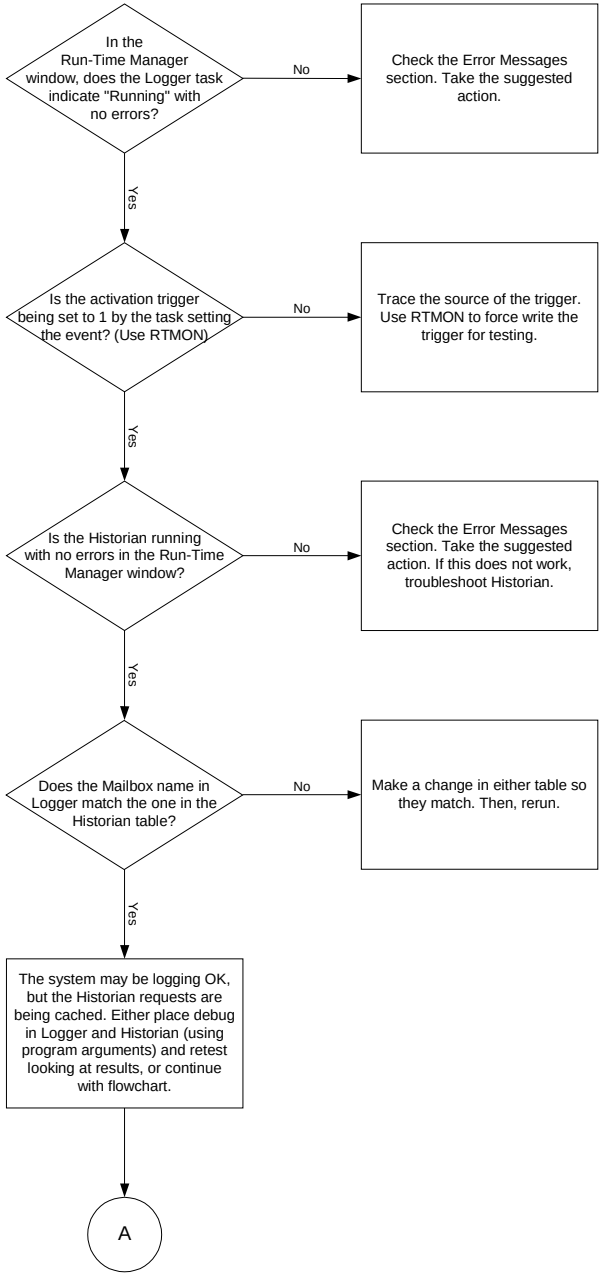
Grouped/ Subgrouped Data

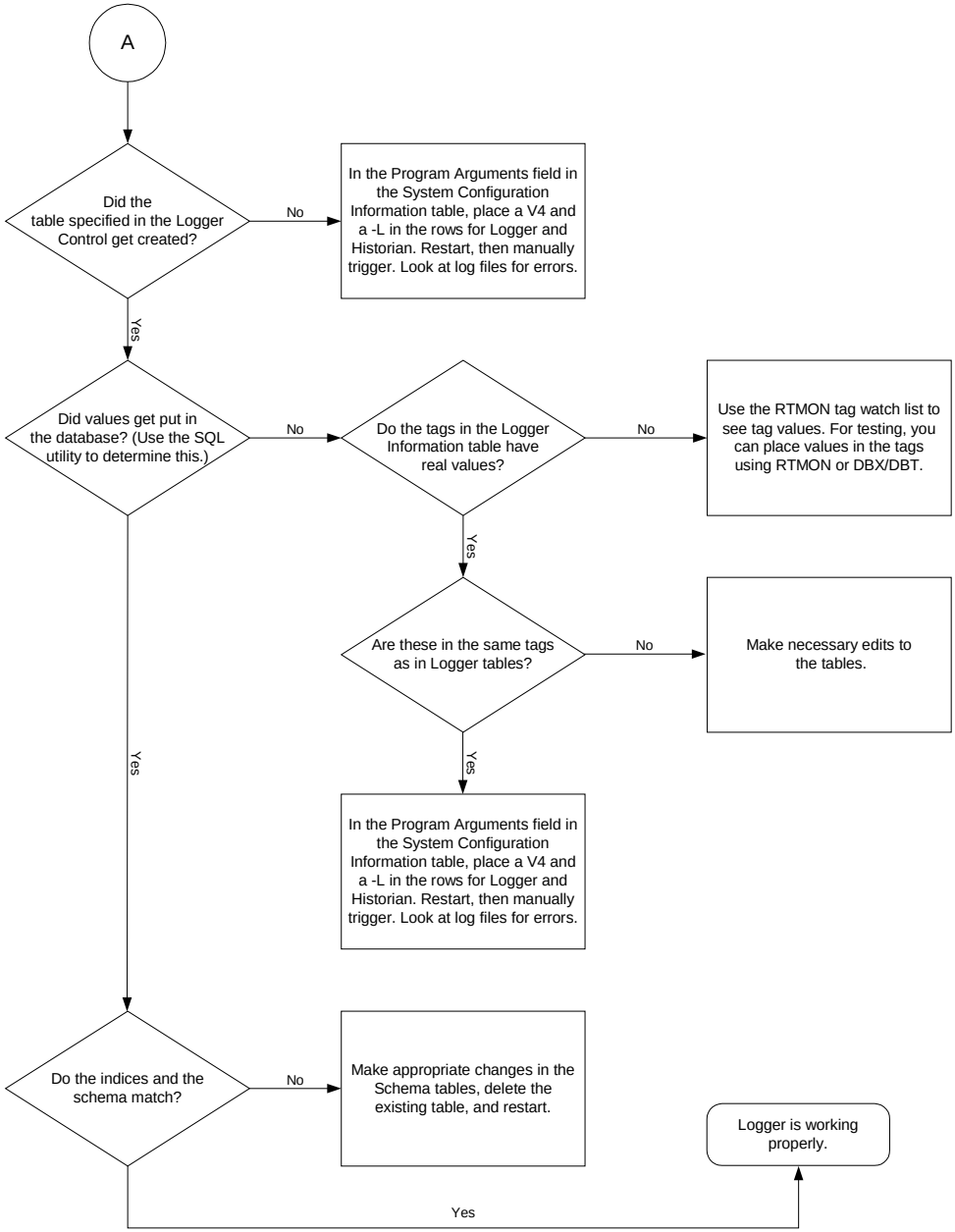


TESTING DATABASE LOGGER OPERATIONS

Once you have configured data logging, test the configuration using RTMON or DBX/DBT to ensure it is functioning properly. Perform the following steps to troubleshoot the logging configuration:

1. Using the Run-Time Monitor (RTMON) or DBX/DBT, add the following items to a watch list for the tags you are logging and triggers you are using for a single logging operation.
2. Using the Tag Input feature, enter sample data into each real-time database tag you are logging and trigger the tag that executes the logging operation you are testing.
3. Check the relational database tables to see if the sample data gets logged.
4. Trigger subgroup rollover and add more sample data to check that subgroup rollover occurs properly.
5. Trigger subgroup rollover using a number that exceeds the maximum number of subgroups allowed to check that subgroup rollover returns to one at the proper time.
6. Trigger group delete to check that group data is deleted at the proper time.


In the Run-Time Manager window, does the Logger task indicate "Running" with no errors?
No
Check the Error Messages section. Take the suggested action.
Yes
Is the activation trigger being set to 1 by the task setting the event? (Use RTMON)
No
Trace the source of the trigger. Use RTMON to force write the trigger for testing.
Yes
Is the Historian running with no errors in the Run-Time Manager window?
No
Check the Error Messages section. Take the suggested action. If this does not work, troubleshoot Historian.
Yes
Does the Mailbox name in Logger match the one in the Historian table?
No
Make a change in either table so they match. Then, rerun.
Yes
The system may be logging OK, but the Historian requests are being cached. Either place debug in Logger and Historian (using program arguments) and retest looking at results, or continue with flowchart.
A


A
Did the table specified in the Logger Control get created?
No
In the Program Arguments field in the System Configuration Information table, place a V4 and a -L in the rows for Logger and Historian. Restart, then manually trigger. Look at log files for errors.
Yes
Did values get put in the database? (Use the SQL utility to determine this.)
No
Do the tags in the Logger Information table have real values?
No
Use the RTMON tag watch list to see tag values. For testing, you can place values in the tags using RTMON or DBX/DBT.
Yes
Are these in the same tags as in Logger tables?
No
Make necessary edits to the tables.
Yes
In the Program Arguments field in the System Configuration Information table, place a V4 and a -L in the rows for Logger and Historian. Restart, then manually trigger. Look at log files for errors.
Yes
Do the indices and the schema match?
No
Make appropriate changes in the Schema tables, delete the existing table, and restart.
Logger is working properly.
Yes

PROGRAM ARGUMENTS

Argument	Description
-D or -d	Enables debug information to be sent to the shared window.
-E or -e	Causes Database Logging to set the completion trigger when the historian task processes the logging operation. By default the completion is set when Database Logging sends the request to the historian mailbox. With this switch, the completion trigger for all log operations means the historian task has processed the logging transaction. Setting the completion trigger does not guarantee the log transaction is successful; it only means the log transaction has completed.
-L or -l	Enables error logging to the log file. By default Database Logging does not log errors.
-Q# or -q#	Sets the maximum number of outstanding asynchronous logging transactions (SQL statements) for the historian task to complete. Once this limit is reached, Logger operates synchronously until the number of uncompleted transactions is reduced. By default Logger allows for up to 100 outstanding logging transactions before operating in a synchronous mode. (# = 100 to 2,000,000,000)
-S# or -s#	Sets the maximum number of concurrently prepared SQL statements active at one time. The default is 30. (# = 1 to 30)
-V1 or -v1	Causes Logger to write the SQL statements generated by the Database Browser to the log file. The Database Browser must have logging enabled for this program switch to work. The default is to not write the SQL statements to the log file.

Argument	Description
-W# or -w#	Sets the maximum time-out in seconds for Database Logging to wait for a response from the historian task. The default is 30 seconds. (# = 5 to 30) For values less than 30 seconds, this switch will only work correctly when the historian initially achieved a successful connection with the database server. If the historian has never successfully connected with the database server, Logger will time out in 30 seconds regardless of this switch setting. The **-W** switch always works for time-outs set at more than 30 seconds, whether the historian initially achieved a successful connection with the database server or not. **Note:** Do not set arbitrarily high values for this argument because it could delay the detection of an actual network or server malfunction.

ERROR MESSAGES

Error Message	Cause and Action
Bad activation trigger in Log name: *logname*	Cause: An invalid activation trigger exists in the DBLOG.CT file. Action: Verify the data type of the tag that triggers the logging operation specified by *logname* is a valid type where *logname* is the name designated for the control entry. Logging operation triggers are specified in the Log Trigger field of the Database Logging Control table. Only digital, analog, long analog and floating-point types are valid. If incorrect, delete the tag, recreate it with a valid data type, and then restart the application.
Bad Completion trigger in Log name: *logname*	Cause: An invalid completion trigger exists in the DBLOG.CT file. Action: Verify the data type of the tag that identifies the completion of the logging operation specified by *logname* is a valid type where *logname* is the name designated for the control entry. Logging completion triggers are specified in the Completion Status Tag field of the Database Logging Control table. Only digital, analog, long analog and floating-point types are valid. If incorrect, delete the tag, recreate it with a valid data type, and then restart the application.
Bad current sequence tag in Log name: *logname*	Cause: The Current Sequence tag has an invalid data type for the logging operation specified by *logname*. Action: Verify the data type for the Current Sequence Tag on the Database Logging Control table is digital, analog, long analog, or floating-point. If incorrect, delete the tag, recreate it with a valid data type, and then restart the application.

Error Message	Cause and Action
Bad current subgroup tag in log name: *logname*	Cause: The Current Subgroup Tag has an invalid data type for the logging operation specified by *logname*. Action: Verify the data type for the Current Subgroup Tag on the Database Logging Control table is digital, analog, long analog, or floating-point. If incorrect, delete the tag, recreate it with a valid data type, and then restart the application.
Bad field type in schema name: *schemaname*, field name *fieldname*	Cause: The Column Type field on the Schema Information table is blank and data is defined in the Column Usage field. Action: Ensure the Column Type field and Column Usage fields of the Schema Information table are compatible.
Bad group delete tag in log name: *logname*	Cause: The Group Delete Tag has an invalid data type for the logging operation specified by *logname*. Action: Verify the data type for the Group Delete Tag on the Database Logging Control table is digital, analog, long analog, floating-point, or message. If incorrect, delete the tag, recreate it with a valid data type, and then restart the application.
Bad group delete trigger in log name: *logname*	Cause: The Group Delete Trigger has an invalid data type for the logging operation specified by *logname*. Action: Verify the data type for the Group Delete Trigger on the Database Logging Control table is digital, analog, long analog, floating-point, or message. If incorrect, delete the tag, recreate it with a valid data type, and then restart the application.

Error Message	Cause and Action
Bad log records in log name: *logname*	Cause: A field name that requires an entry is blank for the logging operation specified by *logname*. Action: Verify all field entries in the Database Logging Control and Database Logging Information tables have valid entries. Cause: The Tag Name field has an invalid data type for the logging operation specified by *logname*. Action: Verify the data type for the Tag Name field on the Database Logging Control table is digital, analog, floating-point, or message. If incorrect, delete the tag, recreate it with a valid data type, and then restart the application.
Bad read tag in log name: *logname*, field name: *fieldname*	Cause: The tag type is invalid for *fieldname*. Action: Verify the data type is valid for *fieldname*. If incorrect, delete the tag, recreate it with a valid data type, and then restart the application.
Bad sequence change tag in log name: *logname*	Cause: The Sequence Change Tag has an invalid data type for the logging operation specified by *logname*. Action: Verify the data type for the Sequence Change Tag field on the Database Logging Control table is digital, analog, long analog or floating-point. If incorrect, delete the tag, recreate it with a valid data type, and then restart the application.
Bad schema records in schema name: *schemaname*	Cause: Unable to read the .CT file for *schemaname*. Action: Verify Schema Name is valid in the Schema Control table and valid information is in the Schema Information table. If it is valid, shut down the Logger task and rebuild the .CT files by forcing a ctgen of dblogger (`ctgen -v2 dblogger`).
Bad send mailbox in log name: *logname*	Cause: The mailbox tag type is invalid for *logname*. Action: Verify the tag type for the Historian Mailbox field in the Database Logging Control table is mailbox. If incorrect, delete the tag, recreate it with a valid data type, and then restart the application.

Error Message	Cause and Action
Bad size for CT # *recordnum*	Cause: A discrepancy exists between the indicated .CT file and the Logger task. Action: Rebuild the .CT file by forcing a ctgen of dblogger (`ctgen -v2 dblogger`).
Bad subgroup change tag in log name: *logname*	Cause: The Subgroup Change Tag is defined with an invalid data type for *logname*. Action: Verify the data type for the Subgroup Change Tag field on the Database Logging Control table is digital, analog, long analog, or floating-point. If incorrect, delete the tag, recreate it with a valid data type, and then restart the application.
Can't close database	Cause: The Logging task issued a **close database** command to the historian task and the historian task could not close the database. The historian may have lost the connection. Action: Verify the historian is connected to the database. Check error messages within the historian. Perform troubleshooting for the historian task. Also for applicable databases, make certain the user is still logged into the database server.
Can't create index: *indexname* for log name: *logname* using schema name: *schemaname*	Cause: An SQL error occurred while creating an index for *logname*. Action: Verify a schema with Schema Name exists in the Schema Control table with a proper index defined in the Index Information table. Create one if one does not exist. If this does not correct the problem, generate a log file by placing a `-v4 -l4` in the System Configuration table for Logging, restart FactoryLink, and reproduce the error condition. Refer to the file FLAPP/FLNAME/FLDOMAIN/FLUSER/LOG/DLMMDDYY.LOG to find all previous errors logged for *logname* and take corrective action as described for each error in this chapter.

Error Message	Cause and Action
Can't create table: *tablename* for log name *logname* using schema name: *schemaname*	Cause: An SQL error occurred while creating *tablename*. Action: Verify a schema with *schemaname* exists in the Schema Control table with a proper index defined in the Index Information table. Create one if one does not exist. If this does not correct the problem, generate a log file by placing a `-v4 -l4` in the System Configuration table for Logging, restart FactoryLink, and reproduce the error condition. Refer to the file FLAPP/FLNAME/FLDOMAIN/FLUSER/LOG/DLMMDDYY.LOG to find all previous errors logged for *logname* and take corrective action as described for each error in this chapter. Also ensure you are connected to a valid database defined in the historian.
Can't log record for log name: *logname*	Cause: A table does not exist for *logname* and no schema is defined. Action: Create a schema using the schema name defined in the Schema Name field of the Database Logging Control table or define a schema name in the existing Schema Name field. Cause: Logging could not prepare an SQL statement to send to the historian. Action: Review the prepared SQL statements in the historian .LOG files. Take corrective action as described for each error.
Can't open CT archive *filename*	Cause: The .CT archive file does not exist or is corrupt. Action: Rebuild the .CT files by forcing a ctgen of dblogger (`ctgen -v2 dblogger`).

Error Message	Cause and Action
Can't open database *database_aliasname*	Cause: Logger issued an **open database** command to the historian and the historian could not open the database. The historian may have lost the connection. Action: Verify the correct database alias name is defined in both the Database Logging Control and the Historian Information tables. For applicable historian tasks verify FactoryLink is connected to the database with a valid logon ID and a password. Generate .LOG files for Logging and the associated historian.
Can't open log file	Cause: Logger cannot open or write to the .LOG file used to debug the task. Action: Verify the FLAPP/FLNAME/FLDOMAIN/FLUSER/LOG directory exists. If not, create it. If the problem persists, remove the -l argument from the Logging task in the System Configuration table. Cause: Logger may have run out of memory or disk space. Action: Verify free memory and sufficient disk space exist on the system.
Can't read CT # *recordnum* in CT archive *filename*	Cause: The archive file is corrupt. Action: Rebuild the .CT files by forcing a ctgen of dblogger (`ctgen -v2 dblogger`).
FactoryLink error filename	Cause: An unknown PAK function error occurred within historian. Action: Contact your technical support representative.
Failed to connect to historian	Cause: An invalid mailbox tag is specified. Action: Verify the tag name in the Historian Mailbox field in the Database Logging Control table is a valid name and the tag type is mailbox. If incorrect, delete the tag and recreate it as defined in the Historian Mailbox Information table. Generate .LOG files for the Logging task and historian by placing a `-v4 -14` in the program arguments in the System Configuration table.

Error Message	Cause and Action
Failed to prepare stmtid	Cause: A nonexistent table name or field name is specified or a syntax error is made in an SQL statement. Action: Verify all entries in the Database Logging configuration table are correct. Rerun the application with a -v4 -l4 as program arguments for the Logging and historian tasks in the System Configuration table.
Historian database error: *filename*	Cause: This message is accompanied by various other messages that describe the cause of the error. Action: Respond to the message displayed on the Run-Time Manager screen or use the .LOG file by rerunning the application with a `-v4 -l4` as program arguments for the logging and historian tasks in the System Configuration table. Refer to the appropriate error message in this chapter for corrective action.
Invalid command line (usage)	Cause: A program argument specified for the Logging task in the System Configuration table is not valid. Action: Verify the specified argument is valid.
Invalid dimension specification in schema name: *schemaname*	Cause: A size is defined for a column that does not match the size of the data being inserted into the column. Action: Increase the value in the Length or Precision field on the Schema Control table.
Invalid global mailbox: DBLOGHISTMBX	Cause: The global mailbox name may be invalid or nonexistent. Action: Verify the tag in the Historian Mailbox field of the Database Logging Control table is a valid mailbox tag. If incorrect or nonexistent, create a new mailbox tag. Only mailbox is an acceptable tag type. Restart the application.

Error Message	Cause and Action
Invalid group configuration. Group tag and current subgroup tag does not exist in log name *logname*	Cause: One of the following tags is missing in the Database Logging Information table for *logname*: Tag Name, Column Usage, or Current Subgroup Tag. Action: Verify each field is defined in the Database Logging Information table. Cause: A column is incorrectly defined with a data type of GROUP. Action: Delete GROUP from the Column Usage field in the Database Logging Information table.
Invalid group configuration. Maximum subgroups defined and current subgroup tag does not exist in log name *logname*	Cause: A maximum number of subgroups is defined, but a Current Subgroup Tag does not exist in the Database Logging Control table for *logname*. Action: Create a Current Subgroup Tag for *logname* or delete the value in the Maximum Subgroups field.
Invalid group configuration. Subgroup change tag exists and current subgroup tag does not exist in log name *logname*	Cause: A Current Subgroup Tag is not defined in the Database Logging Control table, but a Subgroup Change Tag exists for *logname*. Action: Create a Current Subgroup Tag for *logname* or delete the Subgroup Change Tag.
Invalid schema name: *schemaname* in log name: *logname*	Cause: An incorrect or nonexistent schema name is entered in the Schema Name field on the Database Logging Control table. Action: Verify the spelling of the schema name in the Schema Control table or create a schema with the name specified. Restart the application.
Invalid stmtid returned from historian	Cause: The historian shut down. Action: Shut down the logger task and all other historian tasks running. Restart the historian and logger. If this does not resolve the problem, rerun the application with a `-v4 -l4` as program arguments for the logging and historian tasks in the System Configuration table.
Memory error malloc failed	Cause: Insufficient memory is allocated for the Logging task. Action: Check system for free memory.

Error Message	Cause and Action
No activation trigger for log name: *logname*	Cause: A trigger to start the logging operation specified by *logname* is not defined in the Database Logging Control table. Action: Define a trigger in the Log Trigger field for *logname*. Restart the application.
No CTs in CT archive dblog.ct	Cause: Database Logging configuration tables are not set up. Action: Define the Database Logging Control and Database Logging Information tables. Verify the entries are in the correct domain.
No database name in log name: *logname*	Cause: A database alias name is not specified for *logname*. Action: Verify the existence and accuracy of the database alias name in the Database Logging Control table. Change or add as necessary.
No domain name for application directory *directory*	Cause: The domain name is missing or invalid in the application directory. Action: Verify the existence and accuracy of the domain in the operating system. Change or add as necessary.
No field name in log name: *logname*, CT # *recordnum*	Cause: The Column Name field in the Database Logging Information table is blank or misspelled. Action: Open the Database Logging Information table and verify the existence and accuracy of the column name. Change or add a column name as necessary.
No field name in schema name: *schemaname*, CT # *recordnum*	Cause: A column name is not specified in the Schema Information table for *schemaname*. Action: Open the Schema Information table and verify the existence and accuracy of the column name. Change or add as necessary.
No jobs configured in CT archive dblog.ct	Cause: Database Logging configuration tables are not set up. Action: Define the Database Logging Control and Database Logging Information tables. Verify the logging tables are configured in the correct domain.

Error Message	Cause and Action
No mailboxes in CT archive *ctfile*	Cause: A mailbox tag does not exist in the specified archive. Action: Verify a valid mailbox tag is specified in the Logging Control table and the historian tables.
No records in log name: *logname*	Cause: Data is not defined for *logname* in the Database Logging Information table. Action: Open the Database Logging Control table and delete *logname*, or configure a Database Logging Information table for *logname*. Cause: The Tag Name field is left blank in the Database Logging Information table. Action: Open the Database Logging Information table. If the Column Usage field is data, add a valid tag name in the Tag Name field.
No records in schema name: *schemaname*	Cause: Historian is attempting to create a new table and cannot locate a schema. Data from the Schema Information table is not configured or not entered. Action: Open the Schema Control table and configure a Schema Information table for *schemaname*.
No schema defined in CT # *recordnum*	Cause: Data from the Schema Information table is not configured or not entered. Action: Define the Schema Control, Schema Information, and Index Information tables.
No schema name defined in log name: *logname*	Cause: The Schema Name field in the Database Logging Control table is blank or incorrect. Action: Verify the existence of the schema in the Database Schema Creation table. Create it if one does not exist. Action: Verify the accuracy of the schema name. Change or add a valid schema as necessary.
No table name in log name: *logname*	Cause: The Table Name field in the Database Logging Control table is blank or invalid. Action: Verify the existence and accuracy of the table name. Change or add a table name in the Table Name field in the Database Logging Control table as necessary.

Error Message	Cause and Action
Out of RAM	Cause: Database Logging can not acquire memory to run. Action: Restart the task. Install more RAM if the task fails again.
SQL: failed to *historian_function: error code SQL_errornum, SQL_errormsg*	Cause: An SQL error occurred during a logging operation. Historian returned a database-dependent error code. Action: Refer to the user manual for the appropriate relational database software. Consult the error code descriptions and take appropriate action to correct the problem.
SQL: failed to *historian_function: SQL_errormsg*	Cause: An SQL error occurred during a logging operation. Historian returned a database-dependent error code. Action: Refer to the user manual for the appropriate relational database software. Consult the error code descriptions and take appropriate action to correct the problem.
SQL: Log name failed to *historian_function: SQL_errormsg*	Cause: An SQL error occurred during a logging operation. Historian returned a database-dependent error code. Action: Refer to the user manual for the appropriate relational database software. Consult the error code descriptions and take appropriate action to correct the problem.
Task initialization failed	Cause: The Logging task failed to register with the kernel. Action: Verify FactoryLink is up and running by running **flshm -lum**. Also verify it is a purchased option.
Tried to access a nonexistent field	Cause: A field in the database table is missing. Action: Verify the existence of the field in the Database Logging Control table. If it does exist, verify the names match up between the Schema Control and the Database Logging Control tables. Create it if it does not exist. Cause: Permission to access the field is denied. Action: Log in through the appropriate user or have the user grant permission to access the field.

Error Message	Cause and Action
Tried to access a nonexistent table	Cause: The database table does not exist. Action: Verify the table name in the Database Logging Control table. If it does exist, verify the names match up between the Schema Control and the Database Logging Control tables. Create it if it does not exist. Cause: Permission to open the table is denied. Action: Log in through the appropriate user or have the user grant permission to access the table.
Tried to insert a duplicate row	Cause: The user tried to add an existing key value to a unique index. The row can not be inserted causing the data to be lost. Action: Decrease the logging time interval or determine why the associated tag with the index column is not changing.
Unknown function request sent to historian	Cause: An SQL error occurred during a logging operation. Historian returned a database-dependent error code. Action: Refer to the user manual for the appropriate relational database software. Consult the error code descriptions and take appropriate action to correct the problem.
Unsolicited message received from *tagnum*	Cause: A task other than historian wrote to the Logging mailbox. Action: View the X-reference list to determine which task is writing to the Logging mailbox. Change the mailbox tag name of the task writing to the Logging mailbox.

Chapter 7 . . . .

Database Schemas

In FactoryLink, the relational databases are configured in a table format consisting of rows and columns. The schema of the table defines the number, size, and content of the rows and columns.

Schema definitions are created in the Database Schema Creation folder for the Database Logging tables. This folder contains four tables:

- Schema Control table—Assigns unique names to table structures to log data.
- Schema Information table—Defines the columns and table structure attributes.
- Index Information table—Defines which columns the table structure uses as the index. Do not use this table if you are not indexing the table.
- Security Event Logging Schema table—Defines the table columns included in the Security Event Logging table.

Schema Control Table

The Schema Control table sets up the names for each unique data table structure, or *schema*, you create.

Accessing

In your server application, open **Data Logging > Database Schema Creation > Schema Control**.

Field Descriptions

Schema Name — Name that references a unique table structure. Depending upon the application, it might have three unique table structures, as shown in the following examples:

- nongrp_data — For logging nongrouped, nonsequential data. Table content is not grouped or ordered.
- sequence_data — For logging nongrouped, sequenced data. Table content reflects event order or time of occurrence.

group_data — For logging grouped data. Table content is grouped or sorted by a specified criterion.

Valid Entry: up to 19 alphanumeric characters
up to 10 characters for dBASE IV historian

Maximum Records — Number that defines the maximum number of records allowed if you are logging nongrouped data to a dBASE IV historian and want to control the number of records in the tables using this schema. Other database client products such as Oracle or SQL Server require that you perform your own maintenance of the database tables according to your specific needs.

Record rollover occurs when the maximum is reached and starts updating at the location of the first record. This process is repeated in a circular fashion from that point on. DB4_HIST keeps track of only the records added to the database table and not the records deleted from the table.

Leave this field blank to use as many records as your disk space allows or if you are logging grouped data.

Valid Entry: 1 to 9999999999

When designing your application, note the following Maximum Records feature restrictions:

1) Do not delete records from a table for which Maximum Records have been specified.

2) Do not use DBBROWSE positional or logical delete, PowerSQL positional or logical delete or array delete, or DBLOG group delete feature, or BH_SQL delete, or any other delete function. Doing so may result in corrupted data when rollover occurs.

3) Do not browse ALL the records in the table where record rollover is taking place. This can end in DBBROWSE fetching garbage data. This can be avoided if the Maximum Records value is set well in excess of what you need to browse. For example, if you require that your browse operation criteria select the latest 1,000 records logged, set the Maximum Records value at 10,000. This way, the oldest record being overwritten is well removed from the records being browsed.

4) To ensure integrity of the database table, DB4_HIST starts a reindex of the table upon insert of a record with duplicate value in the unique index column. The inserted record is rejected with a “unique key error”; however, the CPU load for reindexing the table might be substantial for large tables. Unfortunately, the unique key error generally occurs at times of heavy CPU usage like application startup, heavy EDI read/write activity, heavy graph swapping activity by operators along with other heavy usage, etc. At such times, there might not be CPU cycles left to update the SECTIME tag for a

few seconds, but DBLOG might go ahead and log this same value to a unique index table column more than once. If you have a very large table and cannot control frequent occurrence of “unique key” errors during your application run, it is strongly recommended that you not use the Maximum Records feature, as it will seriously degrade performance.

5) Due to a problem in the routines used to build this functionality, the integer value “0” for a unique index integer type column is not handled correctly. If you use “0” in your unique index column, you should designate this column of type “char” instead of “integer” or “smallint.” It is not necessary to change the type of the tag that logs to this column or reads from this column, as FactoryLink does data conversion automatically. You should only use unique index column of type integer also if you can guarantee that a value of “0” will never be logged to it. If a value of “0” is logged, at record rollover time the index file will become corrupt and the database useless.

Caution: This applies even if you do not use the Maximum Records feature. Having a “0” in a unique index integer column in your table, your index file will become useless if you ever do a reindex using BH_SQL or DBCHK or any other means in the future.

6) If the records for your different groups are logged non sequentially over time, and you want to delete records based on the age of the groups, do not use the Maximum Records feature. After record rollover, DB4_HIST will overwrite the records starting from the oldest without considering its group ID. You should use the Group Delete feature provided for this purpose. The Group Delete of records and Maximum Records features are incompatible.

SCHEMA INFORMATION TABLE

The schema for each table structure you want to log data to is defined in the Schema Information table. Whether or not the data is grouped or sequenced defines how this table should be completed.

Accessing

In your server application, open **Data Logging > Database Schema Creation > Schema Control > “my schema” > Schema Information**.

Field Descriptions

Column Name Column name. Each name must be unique within a table structure and must conform to the standards for the relational database being used. FactoryLink does not support column names that begin with a number. The size of the column name for dBASE IV is limited to 10 characters. For specific naming conventions allowed, refer to the documentation for the database you are using.

Valid Entry: column name

In the following example for a gasoline station, there are three column names:

group_ID Contains identification assigned to the row of data

order_col Receives the sequence number indicating the order the row of data was logged in.

gals_sold Receives the number of gallons sold.

group_ID	order_col	gals_sold
r87	1	10.5
s93	1	12.2
r87	2	5

Column Usage How information in this column is used, which can be one of the following:

data (Default) Use for columns not used for sequence, time, or group. In this example, **gals_sold** column qualifies for data usage.

sequence Use for columns receiving a sequence number in integer form.

time Use for columns receiving a number in time-stamp form. The time-stamp used is the value from the global tag SECTIME.

group Use for columns receiving the group identification assigned to the data row.

Column Type Keyword that represents the data type contained in the **Column Usage** field. This must be a data type that the relational database receiving the data supports.

Refer to the relational database user guide for the correct syntax.

The following table lists the dependencies.

Column Usage	Column Type
data	All supported types.
sequence	Do not specify a column type.
time	Do not specify a column type.
group	All supported types.

If you do not know the data type when completing this table, specify **unknown** as a placeholder and a reminder to define this data type before completing the configuration. If you do not change **unknown** to a supported type before starting the application, an error occurs.

Valid Entry: keyword name

Valid Data Type: small integer, integer, float, character, date, or number (for the dBASE IV historian)

Default: character

Length or Precision

Maximum number of characters the column can store if you defined a character data type in the **Column Type** field.

If you defined a data type other than character or number, leave this field blank to assume the default 0.

If the **Column Type** field is a number or contains a data type the relational database supports, specify a precision qualifier, such as **xxx** or **xxx,yyy**, where **x** and **y** can be any number. The precision qualifier defines the number of digits (including the decimal) the column allows, and the accuracy of the number before rounding.

Valid Entry: up to 80 alphanumeric characters (cannot exceed 64 if the **Column Usage** field is **group.)**

With the dBASE historian, for a float data type, the maximum precision you can save is a five-digit integer with a precision of five (11,5). A float data type with a value greater than 99,999 is not logged correctly, and is displayed as eleven asterisks. To circumvent this constraint, dBASE users can specify number as the **Column Type**. This allows larger numbers; for example, by using a precision of (13,3), you can log the number 123456789.123.

If Column Type is. . .	Length/Precision is. . .
integer	0 (not applicable)
small integer	0 (not applicable)
float	0 (Maximum float precision in FactoryLink is (11,5), which is the default precision for data values in a float-type column. dBASE IV users: To log values with precision greater than (11,5), use the number data type.
date	0 (not applicable)
character (length)	**xxx** where **x** can be any number. If **Column Usage** is group, the maximum length is 64. Maximum length for dBASE IV is 19 characters.
number (precision)	precision qualifier **xxx,yyy** where **x** and **y** can be any number. **xxx** = total characters, including decimal point, in the value character string and yyy = number of characters at the right of the decimal.

INDEX INFORMATION TABLE

Table indices are defined in the Index Information table. Only complete this table if you are indexing a table. You create an index by specifying the column name or names you want to use as the index in this table. The columns specified are called the index key. You can index on more than one column. Records are retrieved from the database on the first column specified, then the next column, and so on.

In the Schema Index Information table, specify the following information for each index key you want associated with the table structure. You can specify up to 99 different index keys for each schema, although a practical limit is between 6 and 9. Each index key is a separate line item on this table.

Accessing

In your server application, open **Data Logging > Database Schema Creation > Schema Control > "my schema" > Index Information**.

Field Descriptions

Index Nbr Number to uniquely identify the schema index key you are defining. Start with 1 and increment by one for each line item. The more indices, the more disk usage and time it takes to log data.

Valid Entry: index key number between 1 and 99

Unique Index Specify whether or not this index key represents a unique index. The value you enter in this field must be uppercase. This can be one of the following:

YES Duplicate record values are not permitted in the column(s) comprising this index key. If an attempt is made to log a duplicate value, FactoryLink displays an error message and the column of data is not logged.

NO Duplicate record values are permitted in the column(s) comprising this index key.

Column List Column name(s) in the table structure that comprises the index key. The name must match the name in the **Column Name** field of the Schema Information table.

Valid Entry: column name

If you are creating a table using the dBASE IV historian, observe the following constraints:

1. The total number of characters keyed in should not exceed 254, as characters over that limit will not be recognized as valid columns.

2. The width of the columns in bytes (which depends on column type; for example, a column of type char(4) has a width of 4 bytes) should be a number less than the maximum size (100 bytes).

If you are using more than one column as part of the index key, use a plus sign(+) to separate each column name you specify. The multiple columns specified are indexed in the column order they are displayed. For example, given a database table of three columns: Employee Name, City, and Employee Number, you can specify **City+Employee Name**. The data is retrieved alphabetically by City first, and within each city, alphabetically by Employee Name.

OPERATOR EVENT LOG

The Operator Event Log is used to log all changes (events) made by an operator of a FactoryLink application. Whenever the operator performs some action that changes a tag's value from the client project, the OPC Server creates and logs events. Client Builder events, such as connections and disconnections, are also logged. If you are using a third-party OPC client, the log may not reflect the node name of the remote system in the client node database record.

If a new FactoryLink application is created using the Application Setup Wizard or the Create New Application (FLNEW) utility, the client projects have examples for viewing the Operator Event Log found on the RUNMGRS mimic. The simple browser control example shows the database table OPERLOG displayed in reverse TMSTAMP order (latest first).

Configuring Operator Event Log Column Names

The Operator Event Log column names come predefined but can be changed by editing the Security Event Logging Schema table.

Accessing

In your server application, open **Data Logging > Database Schema Creation > Security Event Logging**.

Field Descriptions

Column Alias Internal "alias" name for the type of data logged.

Valid Entry: See table below

Entry	Description
CLIENT	For Client Builder, CLIENT is the node name of the computer where the client is running. For third-party clients, it is the name passed as the client's name.
EVENTMSG	A message associated with the event which is built automatically by the OPC Server. Contains messages in the following fixed formats: • Operator *Name* Logged In • Operator *Name* Logged Out • *Tagname* changed to *newvalue* • Client *NodeName* Connected • Client *NodeName* Disconnected

Entry	Description
EVENTYPE	Type of event, such login, logout, connect, disconnect, update
OLDVALUE	Tag's value immediately before the change
OPERNAME	Name of the operator
OPER2NAME	Second authorization signature
REASON	Operator selected/entered reason
TAGNAME	Name of the tag whose value changed
TMSTAMP	Time that the event occurred
VALUE	Tag's new value after the change

You can change the column names and length in the Security Event Logging Schema table, but the column alias must remain the same. The column order can also be altered from the standard found in the Examples Application and the FLNEW templates.

Configuring an Operator Event Log in an Existing Application

If you have an existing application that needs the Operator Event Log functionality, you need to do the following steps in Configuration Explorer:

1 Add the program arguments to the OPC Server task.

The configuration of the OPC Server task is found in the System Configuration Information table.

Argument	Description
`/OperEvent=mailbox name, database alias name, database table name`	Sets the mailbox name, database alias name, and database table name for the Operator Event Log.
`/OperEvent=OFF`	Turns the Operator Event Log off.

2 Add the mailbox for your historian of choice to the mailbox list. For more information, see the "Historian Mailbox Information Table" on page 267.

3 Set up the database alias in the historian. For more information, see the "Historian Information Table" on page 262. (You can use an alias for a database connection that you have already set up.)

> **Note:** This database table will grow quickly if you have many clients attached to the server. It is recommended that you monitor the size of the table and then archive and purge the table regularly.

Viewing the Operator Event Log

After the Operator Event Log feature is configured in Configuration Explorer, you can view the Operator Event Log using the Database Browser ActiveX control in Client Builder. For more information, see the *Client Builder Help*.

Chapter 8 . . . .

Data Point Logger

The Data Point Logger task logs one data point at a time to a historical database to preserve data for historical purposes through a historian. The historian used for this transfer depends on the relational database receiving the data, such as SQL Server, Oracle, or Sybase.

The Data Point Logger simplifies the task of logging individual data points by providing preconfigured tables. It also allows you to add or remove tags from the list of tags being logged during run time. You can also define your own Data Point Logging tables if you need one other than the pre-defined tables.

Data Point Logging is best for situations when you want to:

- Log a tag only when its value changes
- Use preconfigured tables and eliminate the time spent setting up tables
- Be able to index on log time or tag name or both
- Sort all logs of a tag in order of occurrence
- Configure a tag to be a dynamic pen on a trend chart
- Dynamically change the list of tags being logged during run time

Data Point Logging uses four configuration tables:

- Data Point Logger Control—Relates the Data Point Logging tables to a schema that describes the table parameters.
- Data Point Logging Information—Specifies which tags to log, when to log the tags, and in which table to log the information.
- Data Point Schema Control—Controls the structure or schema of the Data Point Logging tables.
- Dynamic Logging Control—Changes the list of tags being logged during run time without stopping system processing.

DATA POINT LOGGING FUNCTION

Data Point Logging reduces the task of configuring data to be logged. Data Point Logging uses a structured database with the number of columns and the names of the columns predefined by the Data Point Logging Schema table.

Because the table structures are preconfigured, the Data Point Logging task can only be used to log shared, numeric value tags. The tags to be logged can be specified in the Configuration Explorer by means of the Data Point Logging Information table or the Tag Editor.

Preconfigured Data Point Logging Tables

Data Point Logging provides five default Data Point Logging tables, each mapped to a specific default schema name, to provide for efficient storage and retrieval of certain tag types. These log tables are shown in the table below.

Table 8-1 Preconfigured Data Point Logging Tables

Table	Used to Store
FLOATVAL	floating value data
LONGANA	long analog (large integer) data
ANALOG	analog (small integer) data
LOGDATA	general log data
TRENDATA	data that will be used for trend analysis

Each preconfigured Data Point Logging table uses the Database Alias Name **MYDPLOG** which references the relational database where historian sends the data from Data Point Logging. In addition, each default table refers to the Historian Mailbox mailbox tag entry.

Preconfigured Data Point Logging Table Schemas

Data Point Logging provides four default Data Point Logging table schemas, each accepting a different logged data type.

Table 8-2 Default Data Point Logging Table Schemas

Schema Name	Data Type
TAGDATA	float
SMALLINT	smallint (analog, data)
LARGEINT	integer (longana, analog, data)
FLOATVAL	float (float, longana, analog, data)

The maximum number of records allowed in a database table is governed by the relational database being used. For example, the maximum number allowed in a dBASE IV database table governed by any of the four default Data Point Logging table schemas is 1,000,000. Each default schema also specifies a maximum tagname column width of 48.

You must specify a schema for the table in the Data Point Schema Control table if you define your own Data Point Logging table.

Data Logged

Data is logged to a predefined table structure. For each event logged, the database row reflects the following entries.

LOGTIME	TAGNAME	TAGVALUE

LOGTIME	Stores the time the tag was logged
TAGNAME	Tag that was logged at LOGTIME
TAGVALUE	Value of the tag logged at LOGTIME

Only the log time, tag name, and the tag value are recorded in each row. This means less data is stored and captured at each logging trigger, optimizing database storage space.

If a tag is logged more than once during a given second, any values requested to be logged after the first occurrence within that second are ignored.

Logging Methods

With Data Point Logging, you can specify when a tag (data point) is to be logged based on one or more of the following:

- A change in the tag (exception logging)
- A fixed-time interval
- A change in a trigger tag

At task startup, all exception and fixed-time interval tags are logged to create a default beginning reference point. Triggered logging tags are not logged at startup because the trigger tag is not initiated yet.

Logging Based on a Change in the Tag Value (Exception Logging)

You can configure Data Point Logging to log an event whenever the value of the selected tag changes. FactoryLink polls for values of all tags specified for Data Point Logging. If the value has changed from the previous time it was logged, another event is recorded to the log.

If a given tag changes frequently but not all changes are significant, you can configure deadbanding on the tag so only significant changes are logged. This reduces the amount of data logged and decreases system processing time. Deadbanding allows you to specify a band around a tag to determine when the change is significant enough to record the changed value to the system. This band can be an integer or a percentage of the value.

Logging Based on a Fixed-Time Interval

You can configure Data Point Logging to log an event at a specific time interval with the minimum interval being one second. Fixed-time-interval logging rates are based on either seconds, minutes, hours or days, as calculated from a starting point of midnight. It may be appropriate to log a tag value every hour or once a day, depending on the tag.

Fixed-time interval-based logging is tied to time maintained by an internal clock based on the SECTIME global tag. This tag tracks time from the starting point of midnight, January 1, 2004, in intervals of one second.

Logging Based on a Change in a Trigger Tag

You can configure Data Point Logging to log the value of a tag whenever a specified trigger tag is triggered. Whenever the trigger tag is triggered, Data Point Logger logs the event.

LOGGING DATA

Data Point Logger allows you to specify tags to be logged when you are configuring the system and dynamically during run time. When Data Point Logger starts, it looks at several files to determine which one to use to build the log. Data Point Logger determines which of these files is newer:

- The Data Point Logger configuration table file, **{FLAPP}\shared\ct\dplogger.ct**
- The Data Point Save file, **{FLAPP}\log\dplogger.dyn**
- The Data Point Save file specified in the **Command File Tag** field of the Dynamic Logging Control table.

The newer file becomes the list of all tags considered to be configured for logging. If the Command File Tag is not configured or if the tag contains an empty string, the default Data Point Save file is used.

Defining Data to Be Logged During Configuration

When FactoryLink is being configured, the Data Point Logging function can be configured for a specific tag in one of two methods:

- Through the Tag Editor when a new tag is defined
- Through the Data Point Logging Information table

Logging Data Dynamically

You can dynamically add tags to and remove tags from the list of tags to be logged during run-time operation in one or both of the following ways:

- Generate and load a Data Point Save File that contains one or more logging requests
- Enter a single logging request in a tag

Data Point Save File

The Data Point Save File **{FLAPP}\log\dplogger.dyn** contains a list of all tags currently configured for logging. You can create a Data Point Save File that is loaded whenever its associated Read trigger is set. The load process causes the list of tags currently being logged to be overwritten by the list of tags specified in the designated Data Point Save File.

Single Logging Request

Data Point Logging allows you to enter a single logging request by means of the Command Tag defined in the Dynamic Logging Control table. This type of dynamic logging request either adds tags to or removes tags from the list of tags currently configured for logging. Optionally, the logging request can have a tag associated with it that describes the logging request status.

This type of dynamic addition and removal of tags is temporary so, every time Data Point Logging is restarted, the new tag list generated from the Data Point Save File or the Configuration Table file supersedes the existing list.

DATA POINT LOGGER CONTROL TABLE

The Data Point Logger Control table is preconfigured to contain the five default tables and their associated schemas, database alias names, and historian mailboxes.

Accessing

In your server application, open **Data Logging > Data Point Logging > Data Point Logger.**

Field Descriptions

Table Name

Unique name for the Data Point Logger table that receives the data. Each table name must be unique across all relational databases.

If this table does not exist in the relational database, the historian creates it using the schema defined in the Schema Name field when you make a corresponding entry on the Data Point Schema Control table.

Valid Entry: up to 16 alphanumeric characters

Schema Name

Unique name for the table schema that defines the structure of the relational database tables receiving the data. This entry must correspond with an entry in the Data Point Schema Control table.

To revise any Schema Name after you have already logged data to a table using that schema, you must use SQL to drop the table from the list of objects being logged, alter the associated schema, then restart FactoryLink. This recreates the table using the new table structure.

If you are using a database server, such as Oracle or Sybase, alter the table directly using the SQL command ALTER.TABLE. For more information on ALTER.TABLE, see “PowerSQL” on page 411.

Valid Entry: up to 19 alphanumeric characters

Database Alias Name

Name of the relational database where historian sends the data from Data Point Logger. This entry must match a database alias name entry on a database-specific historian table. See “Historians” on page 253 for more information.

Valid Entry: up to 31 characters

Historian Mailbox

Name of the mailbox tag used to transfer data to the historian. This tag must match a mailbox tag entry on a database-specific historian table. See “Historians” on page 253 for more information.

Valid Entry: tag name
Valid Data Type: mailbox

Disable Tag

Tag that disables logging operations.

Valid Entry: tag name
Valid Data Type: digital, analog, float, or longana
Default Entry: digital

DATA POINT LOGGER INFORMATION TABLE

Use the Data Point Logger Information table to relate the tags you are logging to a Data Point Logger table and to configure one or more tags to be logged to a specific relational database table configured in the Data Point Logger Control table. Specify one or more of the following logging methods for each tag:

- On exception—using the Log On Change field.
- Fixed-time interval—using the Log Rate and Log Rate Based On fields.
- Upon an event trigger—using the Log Trigger field.

Ensure the name of the table you want to log data to is displayed in the **Table Name** field at the bottom of the table.

Accessing

In your server application, open **Data Logging > Data Point Logging > Data Point Logger Control > "your log tag name" > Data Point Logger Information.**

Field Descriptions

Log Tag Name of the tag to be logged.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float
Default: digital

Log On Change Whether Data Point Logger is triggered when the value of the tag specified in the Log Tag field changes.

Valid Entry: yes, y, no, n

Log Rate Indicate the interval of time between logging occurrences of the tag being logged. This entry works in conjunction with the time unit the entry in the Log Rate Based On field defines.

Valid Entry: Integer from 1 to 86400

Log Rate Based On Unit of time the Log Rate field entry is based on.

Valid Entry: seconds, minutes, hours, days
Default: seconds

Log Trigger — Name of the tag that triggers the **Log Tag** to be logged.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message
Default: digital

DATA POINT SCHEMA CONTROL TABLE

The Data Point Schema Control table is used to control the structure or schema of the Data Point Logger tables. On the Data Point Schema Control table, you can either:

- Change the storage space occupied by the tag name and tag value fields for the pre-configured Data Point Logger tables.
- Define additional Data Point Logger table schemas. Before using a new schema, however, you must also add a record (row) for it to the Data Point Logger Control table.

Data Point Logger provides four default Data Point logger table schemas, each accepting a different logged data type.

Schema Name	Data Type
TAGDATA	Float
SMALLINT	Smallint (analog, data)
LARGEINT	Integer (longana, analog, data)
FLOATVAL	Float (float, longana, analog, data)

Accessing

In your server application, open **Data Logging > Data Point Logging > Data Point Schema Control.**

Field Descriptions

Schema Name — Unique name for the schema that defines a unique Data Point Logging table structure. This schema is also referenced on the Data Point Logging Control table.

Valid Entry: up to 19 alphanumeric characters

Maximum Records

Maximum number of records allowed in a Data Point Logging table governed by this schema in the dBASE IV database. Only dBASE IV databases use this field. You can use this parameter to automate table maintenance. When the maximum is reached, the oldest record is deleted. If you leave this field blank, as many records as disk space allows are logged.

Valid Entry: 1 to 1,000,000

Tag Name Maximum Width

Maximum width of the Data Point Logging table column receiving the tag name may occupy. If the tag name contains dimension specifications, you must include them when calculating the tag name width. You can use this parameter for efficient data space utilization. If you expect the logged tag name never to exceed 32 characters, then reduce the width to 32.

Valid Entry: 8 to 48
Default: 48

Logged Value Data Type

Data types allowed for the tag being logged to the Data Point Logging table.

Valid Entry: float, integer, smallint or a user-specified, database-dependent numeric data type, such as NUMBER for Oracle

Length or Precision

Specify a precision qualifier, such as *xxx* or *xxx,yyy*, where *x* and *y* can be any number if the Logged Value Data Type field contains a specific data type supported by the relational database. The precision qualifier defines the number of digits and identifies the number of digits allowed in the column and the accuracy of the number before rounding occurs.

PROGRAM ARGUMENTS

Argument	Description
-I	Disable logging tag values at initialization.
-L	Enable logging of SQL statements to a file.
-R<#>	Set maximum number of rows.
-S	Generate Data Point save file after successful dynamic log request.
-T	Generate Data Point save file at task termination.
-V	Enable logging of SQL statements. Statements logged (sent) to Run-Time Manager output window, but not saved.
-W<#>	Set historian time-out parameter. (# = 5 to 30 seconds)
-I	Disable logging tag values at initialization.

ERROR MESSAGES

This section references the directory as FLAPP when discussing errors associated with files in your FactoryLink application directory.

Errors at Task Startup

The following messages report error conditions that occur at Data Point Logging task startup.

Error Message	Cause and Action
Bad size for CT # *record_number*	Cause: (Task terminated.) A discrepancy exists between the indicated .CT file size and the file size specified in the Data Point Logging task. Action: Rebuild the .CT file by forcing a ctgen of the Data Point Logging task using **ctgen -v2 dplogger**.
Can't open CT archive *file_name*	Cause: (Task terminated.) This .CT archive file is missing. Action: Rebuild the .CT file by forcing a ctgen of the Data Point Logging task using **ctgen -v2 dplogger**. Call Customer Support if Data Point Logging does not run.
Can't read CT # *record_number* in CT archive *file_name*	Cause: (Task terminated.) This .CT archive file is corrupt. Action: Rebuild the .CT file by forcing a ctgen of the Data Point Logging task using **ctgen -v2 dplogger**. Call Customer Support if Data Point Logging does not run.
Database alias name not supplied for table name: *table_name*	Cause: This Data Point Logging table does not have a Database Alias Name entry. Action: Enter a valid Database Alias Name.
Historian mailbox not supplied for table name *table_name*	Cause: This Data Point Logging table does not have an Historian Mailbox entry. Cause: Enter a valid Historian Mailbox name.
No CTs in CT archive *file_name*	Cause: (Task terminated.) No configuration data exists in this .CT archive file. Action: Rebuild the .CT file by forcing a ctgen of Data Point Logging using **ctgen -v2 dplogger**. Call Customer Support if the task does not run.

Error Message	Cause and Action
No domain name for application directory *directory_name*	Cause: (Task terminated.) The domain is not defined. Action: Define the environment variable FLDOMAIN.
No jobs configured in CT archive *file_name*	Cause: (Task terminated.) The Data Point Logging Control table must have at least one row of data. Action: Add a row to the Data Point Logging Control table.
Out of RAM	Cause: (Task terminated.) Not enough memory remains to load FactoryLink. Action: Either add more memory or reduce memory consumption.
Task cannot run in domain USER	Cause: (Task terminated.) The User domain is specified. Action: Start the task in the Shared domain.
Task initialization failed	Cause: (Task terminated.) The Data Point Logging task failed to register with the kernel. Action: Verify FactoryLink is up and running by using **flshm -lum**. Also verify the task is a purchased option.
The table *table_name* is not valid	Cause: A Table Name/Schema Name entry in the Data Point Logging Control table does not have a corresponding entry in the Data Point Schema Control table. Action: Enter the missing entry in the Data Point Schema Control table.

Errors at Run Time

Error Message	Cause and Action
Can't open log file	Cause: The error log file could not be opened. Action: Check to see if the disk is full. If so, free up adequate disk space.
Creation of save point file *file_name* failed	Cause: The Data Point save file is not created, possibly because an invalid path is specified in the Command File Tag. Action: Check the syntax of the Command File Tag entry.
Error attaching to database *database_name* <*db_specific_error_message*>	Cause: Depends on content of <*db_specific_error_message*>. Action: Depends on content of <*db_specific_error_message*>.
Error creating table *table_name* <*db_specific_error_message*>	Cause: Depends on content of <*db_specific_error_message*>. Action: Depends on content of <*db_specific_error_message*>.
Error executing cursor for sql statement <*db_specific_error_message*>	Cause: Depends on content of <*db_specific_error_message*>. Action: Depends on content of <*db_specific_error_message*>.
Error initializing cursor for sql statement <*db_specific_error_message*>	Cause: Depends on content of <*db_specific_error_message*>. Action: Depends on content of <*db_specific_error_message*>.
Error occurred during reloading of save point file. Please view log file	Cause: An error occurred while reloading the Data Point save file. Action: Check the error log file.
Error preparing cursor for sql statement <*db_specific_error_message*>	Cause: Depends on content of <*db_specific_error_message*>. Action: Depends on content of <*db_specific_error_message*>.
Global tag *tag_name* not found	Cause: (Task terminated.) Possibly the global tag SECTIME or the global tag DPLOGHISTMBX is missing. Action: Run FLCONV.
Invalid configuration for tag *tag_name* in table *table_name*	Cause: No logging method is specified for this tag. Action: Specify one or more logging methods for the tag.

Error Message	Cause and Action
Invalid syntax in command {LOG|REMOVE}	Cause: Incorrect syntax is used in a LOG or REMOVE statement. Action: Check the command syntax.
Multiple dynamic control records are not supported	Cause: Data Point Logging only reads the first dynamic logging control record. All other rows in this table are ignored. Action: Remove all dynamic logging control records except the first.
Table *table_name* is defined more than once	Cause: (Task terminated.) A table name can only be referenced once in the Data Point Logging Control table. Action: Remove one of the references.
The table *table_name* is not valid	Cause: The table specified in a dynamic logging command does not exist in the Data Point Logging Control table. Action: Enter the name of this table in the Table Name field in the Data Point Logging Control table.
The tag *tag_name* is not valid. Error *<err_msg>*	Cause: An invalid tag name is specified. Action: Specify a valid FactoryLink tag name. *<err_msg>* further describes why the tag is invalid.
Unable to remove tag *tag_name*	Cause: Tag(s) specified in a REMOVE statement cannot be removed. Action: Check the tag name specified in the REMOVE command.
Was not able to read save point file *file_name*	Cause: This Data Point save file was not read, possibly because it does not exist or it exists in a directory other than the default directory. Action: Create one if the Data Point save file does not exist. Specify the full path in the Command File Tag entry if the file exists in a directory other than the default directory.

Chapter 9 . . . .

Event and Interval Timer

Event and Interval Timer allows you to define timed events and time intervals that initiate and control any system function in run-time mode. This task links timed events and intervals to tags used as triggers whenever the event or interval occurs. Timer tags can be referenced by other FactoryLink tasks to trigger some action, such as:

- Read values from a PLC
- Update a report
- Log data to a relational database
- Perform a mathematical procedure

Use this task to signal the occurrence of specified events or intervals by writing to digital tags in the FactoryLink real-time database.

- Timed events occur at a specific time not more than once every 24 hours (for example, Monday at 8:00 A.M.). They are configured in the Event Timer Table.
- Time intervals occur at least once every twenty-four hours at regular intervals of the system clock (for example, every 60 seconds). They are configured in the Interval Timer Table.

OPERATING PRINCIPLES

The Event and Interval Timer task operates in synchronization with the system clock. For each defined interval or event, you must create a digital tag in the real-time database. When the system clock matches the specified event or interval, the task forces the value of this digital tag to 1 (ON).

There is no limit, except the amount of available memory, to the number of event and interval timers that can be defined.

The Event and Interval Timer task also updates global information used by FactoryLink such as the current time, the day of the week, and the month. Such global information is stored in predefined FactoryLink tags, known as reserved tags, each of which is one of the following data types: analog, long analog, or message.

While the Timer task is running, these reserved tags are constantly updated. In order for the Timer task to run, you must have entered an R flag for the Timer task in the System Configuration Table.

The following table lists reserved tags that are updated by the Event and Interval Timer task.

Reserved Tag	Data Type
A_SEC	Analog
A_MIN	Analog
A_HOUR	Analog
A_DAY (Day of month)	Analog
A_MONTH	Analog
A_YEAR	Analog
A_DOW (Day of week)	Analog
A_DOY (Day of year)	Analog
DATE (DOW MM/DD/YYYY)	Message
TIME (HH:MM:SS)	Message
DATETIME (DOW MM/DD/YYYY HH:MM:SS)	Message
YYMMDD (YY-MM-DD)	Message
SECDAY (Seconds since start of current day)	Long Analog
SECYEAR (Seconds since start of current year)	Long Analog
SECTIME (Seconds since January 1, 1980)	Long Analog

CHANGING THE OPERATING SYSTEM DATE AND TIME

To change the operating system date and time while FactoryLink is running, shut down the Timer task, change the date and time, and restart the task. Otherwise, the Timer task tries to catch up by processing missed intervals.

EVENT TIMER INFORMATION TABLE

Use Event Timer to define events that occur at least once every year. For example, use Event Timers to start or stop reports and as triggers to read and write recipe files at a particular time of day.

Accessing

In your server application, open **Timers > Event Timer > Event Timer Information**

Field Descriptions

Tag Name Tag name (example: time8am) to be assigned to the event. When the event occurs, the tag is forced to 1 so its change-status bit is set to 1. The Timer task resets all event timers back to zero at midnight. You can assign more than one tag to the same event.

Valid Entry: tag name
Valid Data Type: digital

Year The 4-digit year the event is to occur (example: 2004). Leave this field blank if the event occurs every year.

Month The month the event is to occur. Can be written numerically (1 to 12) or abbreviated MMM (example: MAR for March). Leave this field blank if the event occurs every month for the selected year.

Day The day the event is to occur. Written numerically (1 to 31). Leave this field blank if the event occurs on every day of the selected period.

DOW Day of the week the event is to occur. Written as the first three letters of the selected weekday (example: MON for Monday). Leave this field blank if the event occurs on every day of the week or only on one specific day.

Hours Hour the event is to occur (0 to 23, with 0 being midnight). The event timer assumes a default value of 0 for blank fields.

Mins. Number of minutes (0 to 59) after the hour the event is to occur. The event timer assumes a default value of 0 for blank fields.

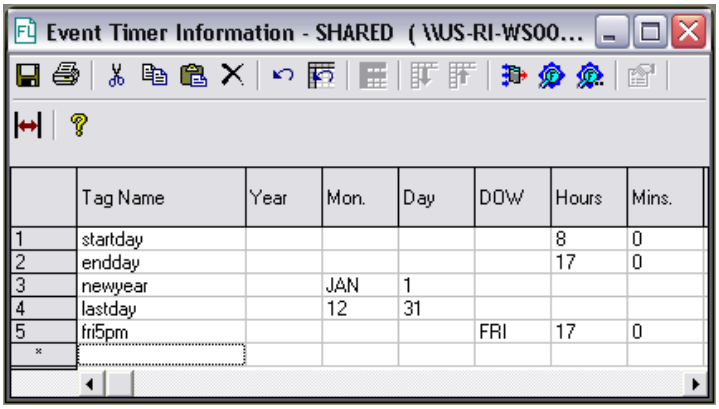
Secs. Number of seconds (0 to 59) after the minute the event is to occur. The event timer assumes a default value of 0 for blank fields.

Note: Between midnight (00:00:00) and the time indicated in the **Hours**, **Mins.**, and **Secs.** fields, the value of the tag an event is linked to is 0 (OFF). The tag value changes to 1 (ON) after the timed event occurs and stays this way until midnight when it changes back to 0 (OFF). Because of this fact, always set a time other than 00:00:00 to avoid the changing back to 0 (OFF) at midnight.

First — Value that determines the action taken upon system startup, if startup occurs after a timed event. Because this field only affects events scheduled for the current date, the system checks the date before changing any values.

- **YES** The tag's value is immediately forced to 1 (ON).
- **NO** Default—the tag's value is left as is and does not change to 1 (ON) until the next occurrence of the timed event.

The Event Timer Information table resembles this example when all information is specified.



	Tag Name	Year	Mon.	Day	DOW	Hours	Mins.
1	startday					8	0
2	endday					17	0
3	newyear		JAN	1			
4	lastday		12	31			
5	fri5pm				FRI	17	0
*							

In this example, the **startday** tag has a value of 0 between midnight and 8:00 A.M. and 1 between 8:00 A.M. and 11:59 P.M. and 59 seconds (23.59.59) each day of the year.

Similarly, the **endday** tag has a value of 0 between midnight and 5:00 P.M and 1 between 5:00 P.M. and 11:59 P.M. and 59 seconds.

The **newyear** tag value has a value of 1 on January 1 of each year and 0 on all other days.

Similarly, the **lastday** tag value has a value of 1 on December 31 of each year and 0 on all other days.

The **fri5pm** tag has a value of 1 each Friday between 5:00 P.M. and 11:59:59 P.M.

INTERVAL TIMER INFORMATION TABLE

Use Interval Timer to trigger events that occur at least once every 24 hours at regular intervals of the system clock, such as every second or every two hours. For example, use interval timers as triggers in Polled Read or Write PLC tables and in Send or Receive tables for FL/LAN Network configurations.

Accessing

In your server application, open **Timers > Interval Timer > Interval Timer Information**.

Field Descriptions

Tag Name Name of the tag (for example, sec5) to be assigned to the interval. You can assign the same interval to more than one tag.

If the tag specified in this field is undefined, the Tag Editor appears when you click **Enter** with a tag type of digital in the **Type** field. Accept this default.

Valid Entry: tag name
Valid Data Type: digital

Hours Indicates the length, in hours, of the interval (0 to 23)

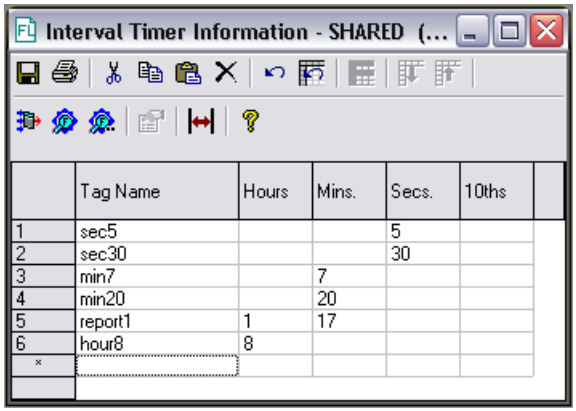
Mins. Indicates the length, in minutes, of the interval (0 to 59)

Secs. Indicates the length, in seconds, of the interval (0 to 59)

10ths Indicates the length, in tenths of a second, of the interval (0 to 9)

Note: The interval timer assumes a default value of 0 if these fields are left blank. At least one of these fields must be filled in with a valid entry. (Not zero, as it is not considered a valid entry.) If the interval can be divided evenly into 24 hours (86400 seconds or 1440 minutes), the timer runs as if it started at midnight. If the interval cannot be evenly divided into 24 hours, the timer starts at system startup.

The Interval Timer Information table resembles the following sample when all information has been specified.



	Tag Name	Hours	Mins.	Secs.	10ths
1	sec5			5	
2	sec30			30	
3	min7		7		
4	min20		20		
5	report1	1	17		
6	hour8	8			
*					

In this example, the **sec5** tag's change-status flags are set to 1 every 5 seconds; that is, when the reserved analog tag A_SEC = 0, 5, 10, 15, ... 55. This timer runs as if it started at midnight; therefore, if system startup time is 9:39:18, the sec5 tag's change-status flags are first set 2 seconds later, at 9:39:20, and every 5 seconds thereafter.

The **sec30** tag's change-status flags are set to 1 every 30 seconds, when A_SEC = 0 and 30. This timer runs as if it started at midnight.

The **min7** tag's change-status flags are set to 1 every 7 minutes after system startup, because 1440 is not evenly divisible by 7.

The **min20** tag's change-status flags are set to 1 every hour, again at 20 minutes after the hour, and again at 40 minutes after the hour.

The **report1** tag's change-status flags are set to 1 every hour and 17 minutes, after system startup.

The **hour8** tag's change-status flags are set to 1 three times a day: at 8:00 A.M., 4:00 P.M., and midnight, regardless of system startup time.

When interval timers are used as triggers for other tasks, such as PLC read triggers or Report Generator triggers, these tasks automatically use the change-status flags associated with these timers.

ERROR MESSAGES

Error Message	Cause and Action
Bad data in ETIMER record *record_number*, file *filename* or **Bad data in ITIMER record *record_number*, file *filename***	Cause: The Event Timer Table contains one or more invalid values. Action: Open the Event Timer Information table. Ensure the correct tag name is entered. Check the data type. Only digital data types are valid on the Event Timer Information table. Choose **View > Object List** to check the data type specified for a tag. Then, use the Search function from the View menu. Choose **View > X-ref List** to display the Cross Reference List to locate all tables that reference a particular tag name. Choose **View > Search**, enter the desired tag name, and click **Enter**.
Bad filename index record in file *filename*	Cause: The TIMER.CT file may not have the correct number of configuration tables archived in it. One or more files may be damaged or missing. Action: Ensure the following files exist: FLAPP/ETIMER.MDX FLAPP/ETIMER.CDB FLAPP/ITIMER.MDX FLAPP/ITIMER.CDB If the files are present and the problem still exists, delete FLAPP/TIMER.CT and restart the application to rebuild the TIMER.CT file.

Error Message	Cause and Action
Can't close file *filename*	Cause: Either the operating system, third-party software, or hardware setup on your system is incorrect or incompatible or an internal error occurred. The FactoryLink real-time database may have been corrupted. Action: Verify the following: The operating system is set up to run FactoryLink (tuning parameters, resources). All third-party software needed by FactoryLink is installed and set up correctly and you have the correct version of FactoryLink. All hardware is correctly set up and all of the hardware is compatible.
Can't find filename in file *filename*	Cause: The TIMER.CT file may not contain the correct number of configuration tables. One of the following files maybe damaged or missing: FLAPP/ETIMER.MDX FLAPP/ETIMER.CDB FLAPP/ITIMER.MDX FLAPP/ITIMER.CDB Action: Ensure the following files exist: FLAPP/ETIMER.MDX FLAPP/ETIMER.CDB FLAPP/ITIMER.MDX FLAPP/ITIMER.CDB If the files are present and the problem still exists, delete FLAPP/TIMER.CT and restart the application to rebuild the TIMER.CT file.

Error Message	Cause and Action
Can't get FactoryLink ID errno *error_number*	Cause: The Timer task is unable to register with the FactoryLink kernel; the Timer task may already be running or more than 31 FactoryLink tasks not including Run-Time Manager may have been started. Action: Check the Run-Time Manager screen to see if the Timer task is running. Also, check the number of tasks that have been started. The maximum number of tasks including Run-Time Manager is 32.
Can't open file *filename*	Cause: The specified file is either nonexistent or opened by another task. Incorrect editing may have caused the problem if the file is exported, edited with a text editor, and then imported. Action: Ensure the specified file name exists. If it exists, check to see whether it was exported, edited with a text editor, and then imported. The file should be unformatted ASCII text.
Invalid tag type for Reserved Timer tag *tag_name*	Cause: The specified reserved Timer tag is the wrong data type. The RTDB may be damaged. Action: Use the Configuration Explorer to correct the reserved timer tag(s).
Read failed for ETIMER record *record_number* in file *filename*	Cause: An I/O error occurred during the reading of an Event Timer.CT record. The TIMER.CT file may not contain the correct number of configuration tables. A file may be damaged or missing. Action: Ensure the following files exist: FLAPP/ETIMER.MDX FLAPP/ETIMER.CDB If the files are present and the problem still exists, delete FLAPP/TIMER.CT and restart the application to rebuild the TIMER.CT file.

Error Message	Cause and Action
Read failed for ITIMER record *record_number* in file *filename*	Cause: An I/O error occurred during the reading of an Interval Timer.CT record. The TIMER.CT file may not have the correct number of configuration tables archived in it. A file may be damaged or missing. Action: Ensure the following files exist: FLAPP/ITIMER.MDX FLAPP/ITIMER.CDB If the files are present and the problem still exists, delete FLAPP/TIMER.CT and restart the application to rebuild the TIMER.CT file.
Reserved Timer tags not defined	Cause: Some or all of the reserved timer tags are not defined. Action: The GLOBAL.CDB and/or GLOBAL.MDX files are damaged. FLNEW did not run. FLNEW creates a blank application structure, including these two files. Part of FLINK/BLANK may be missing.
Wrong Kernel version	Cause: Incompatible versions of FactoryLink software exist on the system. Action: Reinstall the system.
Wrong number of CTs archived in file *filename*	Cause: The TIMER.CT file may not contain the correct number of configuration tables. A may be damaged or missing. Action: Ensure the following files exist: FLAPP/ETIMER.MDX FLAPP/ETIMER.CDB FLAPP/ITIMER.MDX FLAPP/ITIMER.CDB If the files are present and the problem still exists, delete FLAPP/TIMER.CT and restart the application to rebuild the TIMER.CT file.

Chapter 10. . . .

Event Time Manager

The Event Time Manager (ETM) task allows a user to configure objects, functions, and parameters, and to control them based on an Event List that is related to the configuration. An optional user interface can be used to build the Event List independent of the FactoryLink system.

The Event Time Manager was originally available as a third-party option. It is mainly provided now to allow customers who used it in the past to upgrade their systems to the latest version of FactoryLink.

ETM contains the following features:

- Virtually unlimited number of tags in list of time-controlled objects, functions, and events.
- Enabling/disabling control per object, Min/Max control, and standard value.
- Configurable functions with actions SET, TGL, ON, OFF, ADD, SUB.
- Automatic adjustment of time including daylight savings.
- Configurable special events or periods for Easter, Christmas, and so on.
- Events configurable as repetitive and with period limitation, such as always on Monday 12:00 from May to September.
- Configuration check by running in test mode.
- Support of several file formats for the Event List several file formats. The ASCII file formats (Csv, Cat, Txt, WoA) are read directly from the file system and the database format (dBaseIV) is read using the FactoryLink Browser.
- An optional ETM Input Mask program allows for on-line configuration of events by using predefined tables (input masks).

OPERATING PRINCIPLES

This section describes various operating principles and concepts associated with the ETM task and the ETM Input Mask program.

Event Definition/Special Events

Every event object is associated with a function (representing a class) and its commands. When defining objects you tell ETM which tags it must control and the associated function defines how ETM can control those object. By defining events, you tell ETM when it has to process an action.

The location for ASCII Event Lists is %flapp%\ETM; for historian files it is freely selectable.

An event is defined by fields Fix Date, Event Time, Weekday, Special Event, Valid from..through.

- The date/time format is ISO 8601 and starts with the FactoryLink time calculation (1980-01-01 00:00:00).
- The date (YYYY-MM-DD) is defined by year (4 digits), month (2 digits) and day (2 digits) separated by a hyphen or minus sign {-}.
- The time (hh:mm:ss) is defined by hours, minutes and seconds (each of 2 digits) separated by a colon {:}; the time resolution is one second.
- The day begins at 00:00:00 and ends at 23:59:59.
- The fields Valid from...through require the date format and are used to limit the span of a repetition.

Weekday and Special Event are further possibilities to describe an event. You can specify the available entries in the ETM Runtime Parameter table. Every entry can be negated by a preceding hyphen {-}.

An event is defined exactly for one time (explicit) or as a repetition, being processed more than once. On a repeated event at least one field in the date and time string is empty. Preceding delimiters must be declared. An empty field generally means *always* whereas Weekday and Special Event are considered one field. An empty time field means at *00:00:00*. As a logical rule consider an event to occur at the time: [Weekday **OR** Special Event] **AND** [Fix Date] **AND** [Valid from..through] **AND** [Event Time].

The examples in the following tables illustrate explicit events and repetitions.

Table 10-1 Explicit Events

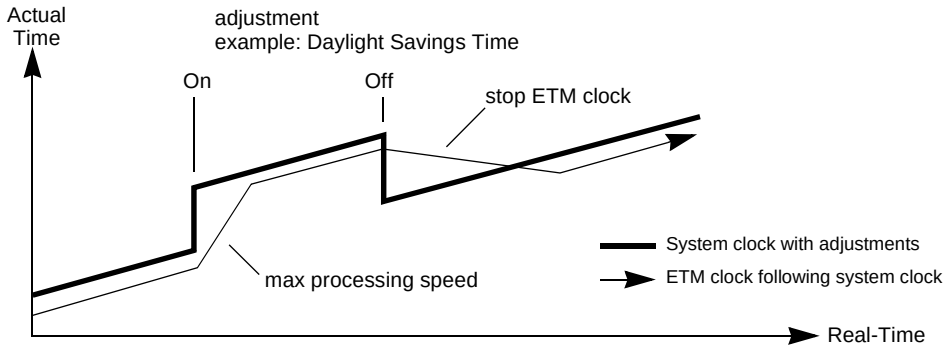
Week Day(s)	Special Event(s)	Fix Date	Valid from	Valid through	Event Time	**Remark** (Date/Time format is ISO 8601)
		2000-03-23			14:05:00	
		2004-01-01			00:00:00	
		2004-01-01				Same as line above, since an empty Event Time means 00:00:00
		2005-01-01				Same as line above, but 1 year later

Table 10-2 Repetitions

Week Day(s)	Special Event(s)	Fix Date	Valid from	Valid through	Event Time	**Remark** (Date/Time format is ISO 8601)
Mon			-03	-10	08	Always every Monday at 8 o'clock from March through October
Mon	-Easter -Xmas				8	Every Monday at 8 o'clock but not at Easter and Christmas
Sun	Easter Xmas				8	Every Sunday or every day at Easter or Christmas at 8 o'clock
		-01-01	2003	2007		From 2003 through 2007 on every January 1st at midnight (00:00:00)
		2001- -01			:30	On the first of every month in 2001 always at half past
						Every day at midnight
		- -				Every day at midnight
Mon Wed Fri					12	Always at 12 o'clock on Monday, Wednesday and Friday
-Sat -Sun					20:00:00	Every day at 20 o'clock but not on Saturday and Sunday

Time Mechanism

The internal clock (ETMClock) follows the system clock in one second intervals. The ETMClock is only incremented if the system clock precedes it. If the ETMClock is more than one second behind, ETM runs with a shorter interval to make up leeway. The short interval can be adjusted by a program argument. In this way events will always be processed even if the system clock is changed (daylight saving) or the CPU is blocked by other tasks. Furthermore a burst of events may be processed in batches without overloading the system.



Event List and Buffer Handling

The Event List is read periodically (asynchronously) by an adjustable interval. Every repetition and every explicit event from the actual time until the end of the day is read from the Event List. A buffer time can be specified to increase the span of explicit events read. Reading the file and preparing the data for processing can consume time depending on the file size. But until data is completely prepared, ETM still runs with the list read one cycle before. In case of unsuccessful Event List, reading or preparing the buffer read one cycle before guarantees that no event is lost. Modifications on the Event List only take effect on successful reading. If your modifications will be available immediately, you can force reading the event list database by triggering a tag configured in the ETM Runtime Parameter table.

Run-time and Test Operation Mechanism

When the ETM run-time task is starting, it runs automatically in the normal run-time mode. It can also run in a test mode to check the commands and the event list. In this test mode, ETM can be initialized on a certain date and time and process the event list speeded up for that day, such as to test the events released later at Christmas.

Caution: This may cause unpredictable reactions. For example, telling ETM to stop the heating system, it is best to cut off the communication to the PLC first.

Mode Description

The operation mode is available as an input/output tag configured by Parameter tag OpMode in the ETM Runtime Parameter table. The tag is subdivided like a bit field in command and information modes. The user can set ETM to a certain mode by forcing the tag with a command mode. ETM always shows its actual state with information modes. The following modes are available:

Mode	Dec Hex Value	Description
Auto	00x0000	Command and information mode, the internal clock follows the system clock
Test	10x0001	Command and information mode, the internal clock follows the value of the Parameter tag ExternalTime configured in the ETM Runtime Parameter table
Off	20x0002	Command and information mode stop the internal clock
Init	256 0x0100	Information mode, ETM is processing an initialization routine as at ETM startup
ReadDB	512 0x0200	Information mode, ETM is reading the Event List database

Mode Transition Description

Auto

After each SleepTime (Program Argument), ETM checks if the system clock has changed and if the internal clock is late. If so, ETM processes the events according to the faster internal clock. Then ETM suspends again for the duration of SleepTime.

If the ReadPeriod (Program Argument) has expired, ETM goes to ReadDB to update the Event List and returns to its previous state.

If Parameter <OpMode> is set to 2, ETM goes to Off and stops processing. The Parameter <ExternalTime> is initialized with the actual value of the system clock.

If the <OpMode> is set to 1, ETM goes Test/Init. The <ExternalTime> is initialized with the actual value of the system clock; the Event List is read and an initialization on that <ExternalTime> is started.

Auto/Init

ETM reads the Event List database. If the initialization is completely processed, ETM goes to Auto.

If the <OpMode> is set to 2, ETM goes to Off mode and stops processing. The <ExternalTime> is initialized with the actual value of the system clock.

Test/Init

ETM reads the Event List database. If the initialization is completely processed, ETM goes to Test mode.

If the <OpMode> is set to 2, ETM goes to Off mode and stops processing.

Off

If the <OpMode> is set to 0, ETM goes to Auto/Init mode. The Event List is read and an initialization on the system clock is started.

If the <OpMode> is set to 1, ETM goes to Test/Init mode. The Event List is read and an initialization on the actual value of <ExternalTime> is started.

Test

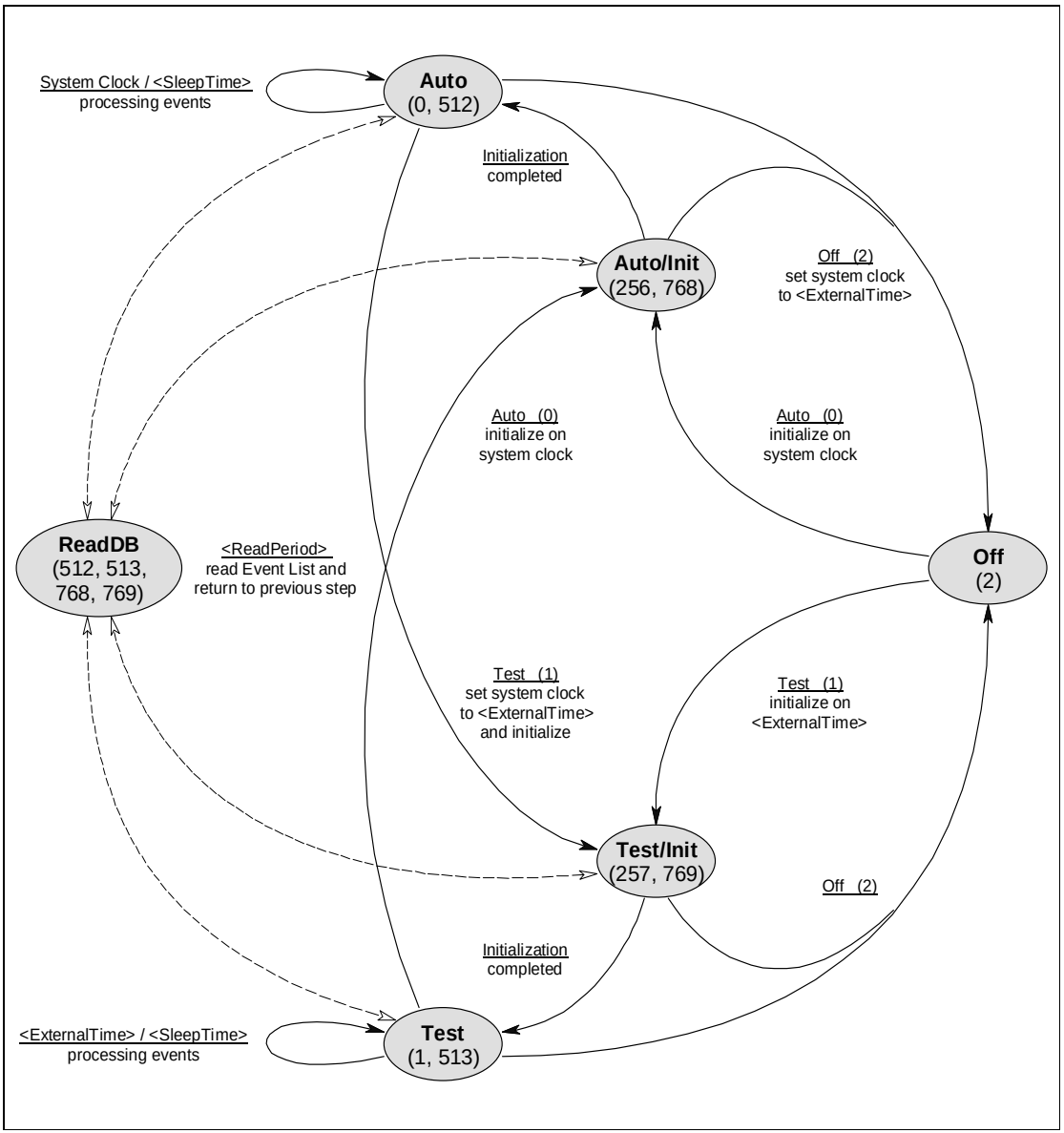
After each SleepTime, ETM checks if the Parameter <ExternalTime> tags value has changed and the internal clock is late. If so, ETM processes the events for the increased internal clock. Then ETM suspends again for the duration of SleepTime.

If the ReadPeriod has expired, ETM reads the Event List database again.

If the <OpMode> is set to 2, ETM goes to Off and stops processing.

If the <OpMode> is set to 0, ETM goes to Auto/Init. The Event List is read and an initialization on the system clock is started.

Figure 10-1 Mode Transition Diagram



ETM Input Mask Program

ETM supports an optional ETM Input Mask program that allows on-line configuration of events using predefined tables (input masks). The ETM Input Masks are automatically linked to the Objects, Functions, Weekdays, and Special Events previously configured in the FactoryLink application. When the ETM Input Mask program is closed, a modified Event List is stored in the FactoryLink application so the ETM Task can read the list for online processing of the new events. For valid Event List Formats, see the ETM Parameter table.

ETM Object Information Table

The ETM Object Information Table identifies objects that ETM controls. Every object can be disabled at run time and can have its own value range, default value, unit, and description.

Accessing

In your server application, open **Other Tasks > ETM Event Timer Manager > ETM Object Information**.

Field Descriptions

Field Name	Description	Valid Entries	Valid Database Types
Object Control Tag	Tag representing the object to be controlled. A function having one or several commands must be associated with the object. Depending on the command and the specified event, ETM will write to this tag.	(optional, case sensitive, 48 characters): any valid tag name	DIGITAL, ANALOG, LONGANA, FLOAT or MESSAGE
Enable Tag	Tag in combination with the contents of the *Enable Value Tag to interlock events on this object. If the condition given by these two fields is false the events are ignored, otherwise they are enabled. If either field is empty no event will be ignored.	(optional, case sensitive, 48 characters): any valid tag name	DIGITAL, ANALOG, LONGANA, FLOAT or MESSAGE

Field Name	Description	Valid Entries	Valid Database Types
*Enable Value Tag	Tag or character constant in combination with the contents of the Enable Tag to interlock events on this object. If the condition given by these two fields is false, the events are ignored; otherwise they are enabled. If either field is empty no event will be ignored. If the two fields have different tag types the *Enable Value Tag is transformed to the type of the Enable Tag before the test will be made. At the beginning of the contents, a logical operator can be assigned to determine the condition. The following operators are allowed: = Equal (default if no operator given) != NOT equal < Less than <= Less or equal > Greater than >= Greater or equal The following two examples of character constants represent the same condition and will disable every event on an object if the contents of the enable tag is not 3: ‘3 ‘=3	(optional, case sensitive, 48 characters): any valid tag name or character constant	DIGITAL, ANALOG, LONGANA, FLOAT or MESSAGE
Function	A function must be associated with every object. It describes the class or type of the object. Functions can have a number of commands as described for the ETM Function Information Table.	Valid entries (required, case sensitive, 23 characters): any function name referencing commands in the appropriate Function Information table.	
*Min Value	Tag or number constant representing the lower limit of the object value. If a command of an event forces the objects value below this limit, the value will be corrected to the contents of this field.	(optional, case sensitive, 48 characters): any valid tag name or number constant	ANALOG, LONGANA or FLOAT

Field Name	Description	Valid Entries	Valid Database Types
*Max Value	Tag or number constant representing the upper limit of the object value. If a command of an event forces the objects value above this limit, the value will be corrected to the contents of this field.	(optional, case sensitive, 48 characters): any valid tag name or number constant	ANALOG, LONGANA or FLOAT
*Standard Value	Tag or character constant representing the default value of the object if no other value is specified for an event.	(optional, case sensitive, 48 characters): any valid tag name or character constant.	DIGITAL, ANALOG, LONGANA, FLOAT or MESSAGE
Unit	Individual description of the unit of the objects value displayed in the ETM Input Masks	(optional, case sensitive, 23 characters): any	
Object Control Tag Description	FactoryLink Tag Description of the Object Control Tag displayed in the ETM Input Masks.	(optional, case sensitive, 80 characters): the description is entered at Tag specification and cannot be modified in this column	

ETM Function Information Table

The ETM Function Information Table identifies the commands for a particular class of objects (typical object). Every object is derived from a class and the commands describe what you can do with an object of this class.

> **Note:** The Function name (Class) is defined in the ETM Object Information table and displayed in the Input Masks and Select List of the ETM Input Mask add-on program.

Accessing

In your server application, open **Other Tasks > ETM Event Timer Manager > ETM Object Information > "my ETM" > ETM Function Information**.

Field Descriptions

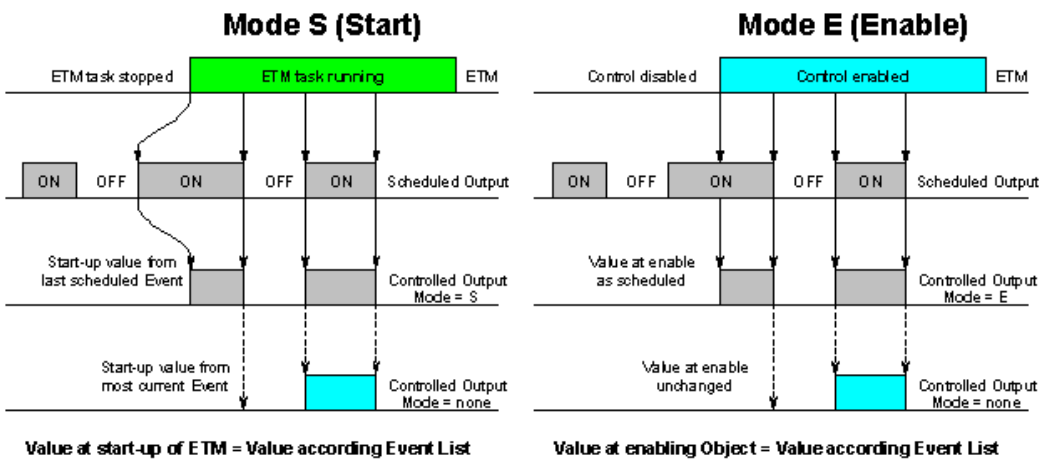
Field Name	Description	Valid Entries	Valid Database Types
Command	Individual description of the command. The descriptions are used in the Event List and displayed in the ETM Input Masks. By entering a command in the Event List or in the ETM Input Masks, ETM executes the appropriate action and processes the event with the *Standard Value or with the higher ranking *Preset Value. Or, on the special command "?" the user can enter a value that even supersedes the ***Preset Value**.	(required, case sensitive, 23 characters): any name or „?“ for user specified input value	
*Preset Value	Tag or character constant representing the value of the command if no other value supersedes it. If specified, this field supersedes the *Standard Value.	(optional, case sensitive, 48 characters): any valid tag name or character constant	DIGITAL, ANALOG, LONGANA, FLOAT or MESSAGE

Field Name	Description	Valid Entries	Valid Database Types
Action	Standard action applied on the value for this command. The following actions are available, enter a name or select one from the list: ON - The object control tag will be set to ON (value 1). OFF - The object control tag will be set to OFF (value 0). SET - The object control tag will be set to the value valid for this command. ADD - The value valid for this command will be added to the current control tags value. SUB - The value valid for this command will be subtracted to the current control tags value. TGL - The object control tag will be set depending on the actual value (if the value is 0 set to 1, if value differs from 0 set to 0)	(required, case insensitive, 7 characters): any valid action, default is SET.	
Mode	Flags to modify a command with regard to **E**nable, **I**nterlock and **S**tartup behavior. See the table below for valid modes.	(optional, case insensitive, 7 characters): any valid mode or none (default)	

The following table provides the valid Mode entries.

Mode	Description
E	Store and forward this command until the control tag is enabled. Processing of this command can be delayed if it was disabled at the last proposed event time. However, it will be abandoned if superseded by any other command appearing as a possible event prior to enabling. This mode is used to shift a command using the enable tag; for example, the command to turn off the lights at 7:15 pm can be disabled only after enabling at 10:20 pm.
I	This command is processed at the proposed event time regardless of its actual interlock. This mode is used to unconditionally force the control tags value, such as to always turn off the lights at 2:30 am.
S	This command is processed at ETM start-up to initialize the Control Tags value. The last command prior to start-up (first one found backwards in the Event List) with this mode is applied. You can set the span (event history time) to scan backwards from start-up for finding commands with this mode (Program Argument – StartupTime#). For example, mode S can be used to turn on the lights later, if the system was down at the actual event time.
SE	This command combines mode S and E. It is processed when ETM is started to initialize the control tags value, and a possible disabled command is stored and forwarded until the control tag is enabled.
SI	This command combines mode S and I. It is processed when ETM is started to initialize the control tags value, and it is processed at the proposed event time regardless of its actual interlock.

The following graphic illustrates the behavior of Startup and Enable mode.



ETM Runtime Parameter Table

The ETM Runtime Parameter table allows to determine the Weekday names in a useful manner, to specify objects with a special meaning (priority processed) or to set the information about the Event List Format.

> **Note:** Parameters for week days and for the special event can be listed several times in order to suit for multiple abbreviations and events, respectively.

Accessing

In your server application, open **Other Tasks > ETM Event Time Manager > ETM Runtime Parameter**.

Field Descriptions

Field Name	Description	Valid Entries	Valid Database Types
Parameter Name	This column is used to select any desired <Parameter> as described below. Enter a <Parameter> or select one from the list.	(optional, case sensitive, 15 characters): any valid Parameter name	
Parameter Tag	For some parameters a tag is applicable to fulfill the functionality.	Depending on Parameter, any valid tag name	Depends on the Parameter
Parameter Argument	For some parameters an argument is applicable to fulfill the functionality. See the table below for valid parameters.	Depending on the Parameter up to 79 char.	

Table 10-3 lists valid parameters for the Parameter Argument field.

Table 10-3 Valid Parameters for ETM Runtime Parameters Screen

Parameter	Description	Valid Entries	
List Format		(required, case insensitive, 23 characters): any name or „?“ for user specified input value. Valid entries are listed below:	
		db4	dBaseIV (CodeBase 4.5); read Event List through Database historian and Database Browser; Event List location and file name are freely configurable in historian.
		Cat	**Default Format.** Read Event List directly; field size fixed, field separator {~}, no string indicator; Event List location and file name must be **{flapp}/etm/etm_list.dat**.
		Csv	Read Event List directly; field size floating, field separator {;}, string indicator {"} (optional); Event List location and file name must be {flapp}/etm/etm_list.csv.
		Txt	Read Event List directly; field size floating, field separator TAB, no string indicator; Event List location and file name must be {flapp}/etm/etm_list.txt.
		WoA	Read Event List directly; field size floating, field separator {,}, string indicator {"}; Event List location and file name must be {flapp}/etm/etm_list

Table 10-3 Valid Parameters for ETM Runtime Parameters Screen (continued)

Parameter	Description	Valid Entries
List DBBCtName	The argument column (case insensitive) specifies the corresponding ct name if the Event List is read by the Database Browser.	Default argument is ETM_EVENTLIST.
SpecialEvent	Special Events are identified by controlling the appropriate tags of type DIGITAL, ANALOG, LONGANA, FLOAT or MESSAGE in the Parameter tag column. This can be done by ETM or any other task. A Special Event is active as long as the tags value differs from 0. It is handled with higher priority if two events occur at the same time. A unique entry in the argument column (case sensitive) identifies its reference name in the Event List or in the Input Masks. A Special Event can be used as a condition for other events in the Event List.	
ReadDB	The Event List is parsed periodically. To force reading, this trigger can be set. It is specified by an entry in the Parameter tag column (type DIGITAL, ANALOG, LONGANA or FLOAT).	
InternalTime	Information on the actual internal time of ETM. The tag in the Parameter tag column is updated periodically (normally 10 s, while testing 1 s). The displayed format is FactoryLink SECTIME for type LONGANA or ISO 8601 YYYY-MM-DD hh:mm:ss for type MESSAGE.	
ExternalTime	This is the timer that ETM will follow in the test mode instead of the system clock. The tag in the Parameter tag column must be set in the format of FactoryLink SECTIME for type LONGANA or ISO 8601 YYYY-MM-DD hh:mm:ss for type MESSAGE.	

Table 10-3 Valid Parameters for ETM Runtime Parameters Screen (continued)

Parameter	Description	Valid Entries
Clock Tick	Normally ETM runs synchronous to the system clock. If ETM is more than 1 s behind, it takes the interval time [ms] set to the tag (type ANALOG) in the Parameter tag column until ETM has caught up the system clock. With this parameter, the user can at run time control ETM so as to catch up time (clock tick value < 1000 ms) or run delayed (clock tick value > 1000 ms). The configuration of this Parameter supersedes the program argument used for the same effect.	
OpMode	Tag (type LONGANA) in the Parameter tag column used to set / display the operation mode of ETM.	See “Operational Modes” on page 188.
Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday	The entry in the program argument column identifies this weekday. It is used in the Event List and in the Input Masks. This Parameter can be listed several ways, such as Mon, Mo, MO, Lun, to suit multiple abbreviations or languages.	Case-sensitive weekday entry
Workday	Same as <Monday>..<Sunday>. This Parameter can be listed several ways, such as Workday, WRK. With this setting, the user can control the span of the workday. If no tag is specified, ETM identifies the time span from Monday 00:00:00 to Friday 23:59:59 as workday.	Optional tag (type DIGITAL, ANALOG, LONGANA, FLOAT or MESSAGE)
Weekend	Same as Monday. This Parameter may be listed several ways, such as Weekend, WND. With this setting, the user can control the span of the weekend. If no tag is specified, ETM identifies the time span from Saturday 00:00:00 to Sunday 23:59:59 as weekend.	Optional tag (type DIGITAL, ANALOG, LONGANA, FLOAT or MESSAGE)

Operational Modes

The following operational modes can be set by the user to force ETM in the desired mode and ETM will use them to display the mode it is running:

Mode	Dec HexaDec	Description
Auto	0 0x0000	ETM follows the system clock
Test	1 0x0001	ETM follows the <ExternalTime> tags value
Off	2 0x0002	Stop the internal clock

The following operational modes can be displayed in addition to ones above:

Init	256 0x0100	Processing an initialization routine (same as required on start up)
ReadDB	512 0x0200	Reading the Event List database

The following examples show the possible combinations:

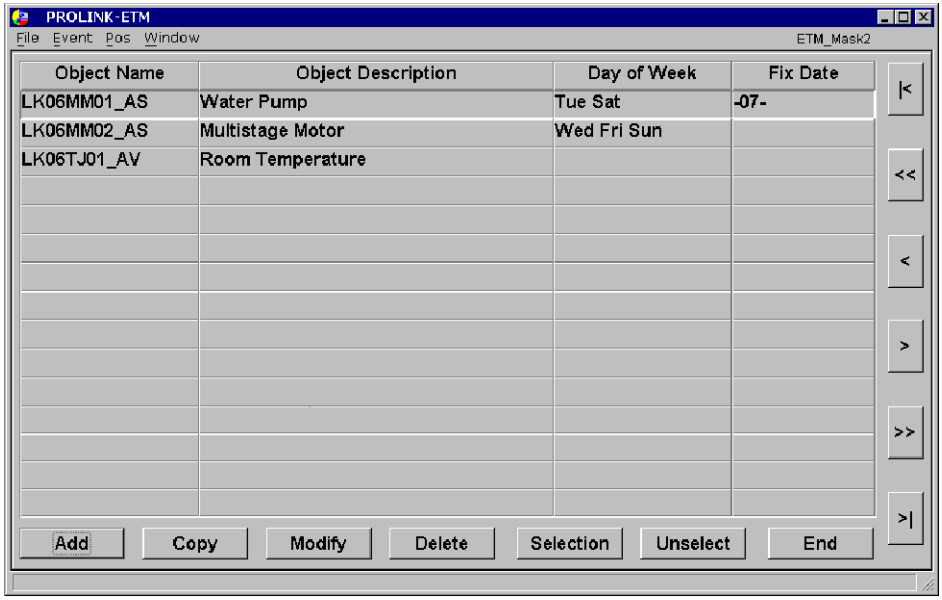
Auto / Init	256 0x0100	Initializing for Auto mode
Test / Init	257 0x0101	Initializing for Test mode
Auto / ReadDB	512 0x0200	Reading the Event List while running in Auto mode
Test / ReadDB	513 0x0201	Reading the Event List while running in Test mode
Auto / Init / ReadDB	768 0x0300	Reading the Event List while initializing for Auto mode
Test / Init / ReadDB	769 0x0301	Reading the Event List while initializing for Test mode

ETM EVENT CONFIGURATION MASKS

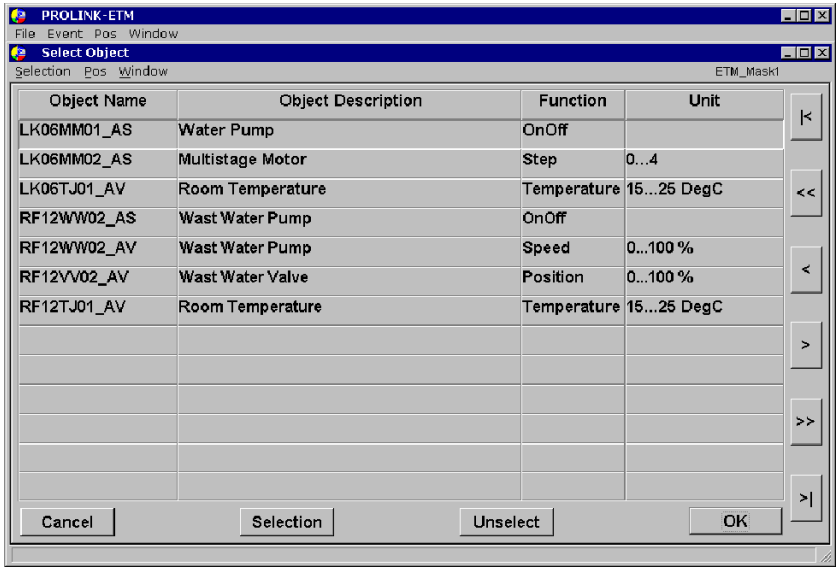
The ETM Input Mask Program can be started by double-clicking the specific icon on your desktop.

Event Overview and Select List

The startup screen shows an overview of all objects in alphabetical order of the defined events. The following window shows all weekly events after filling in the input masks:



The following screen shows all specified objects after clicking Choice on any menu.



All functions can be accessed by mouse, Tab and Enter key, or by selecting an item from the menu. To copy, modify or delete a record, the desired object must be selected prior to releasing the function. For copy and modify, open the Event Configuration Mask to define the events for the selected object.

Scroll Buttons are on the right hand side of the list:

|<	Go to the first page
<<	Page up
<	Line up
>	Line down
>>	Page down
>|	Go to the last page

The following main functions are available:

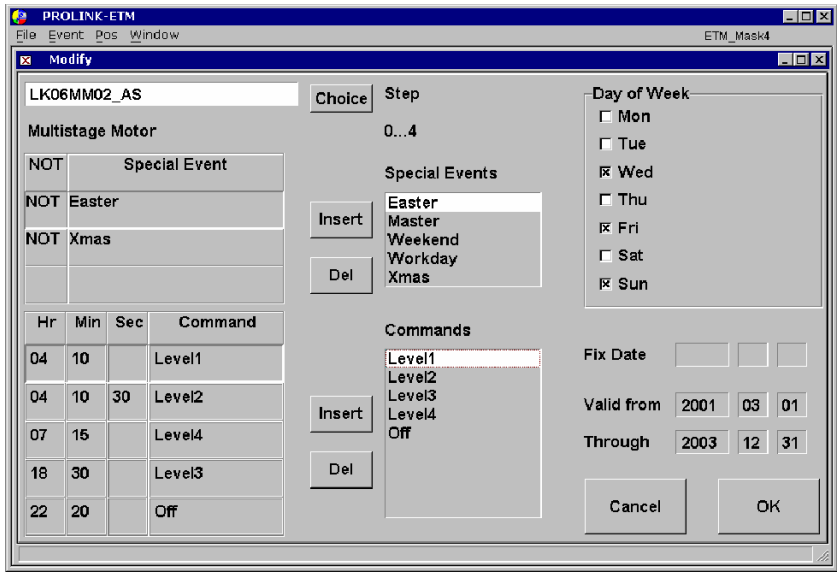
Add	Select a new object and define events for it.
Copy	Copy the events of the selected object to another one and modify it.
Modify	Modify the selected object.

Delete	Delete the selected object and its events. You will be prompted to acknowledge deletion.
Selection	Specify a list of selected objects matching user defined criterion (filter). A window displays where the desired criterion can be defined. After defining the criterion and clicking OK, the Select List displays.
Unselect	Cancel any selection and clear the criterion entered before. A list of all objects will display again.
End	Shutdown the ETM Input Mask program. If changes were made, a confirmation message appears to update the Event List. Click Yes to store the information, No to ignore the modifications, or Cancel to not end the session.

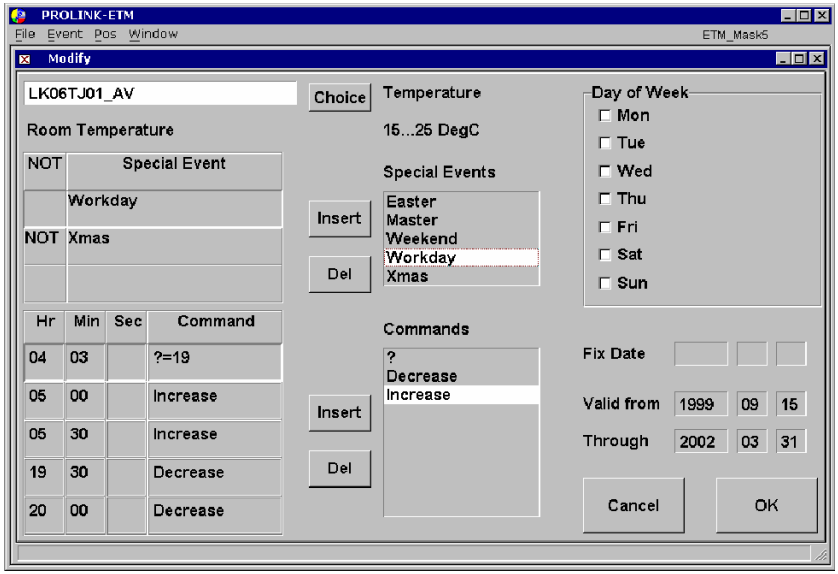
Event Configuration Input Mask

For functions Copy and Modify the fields in the Event Configuration Mask will appear with data already existing, for function Add they will be set default (normally empty).

The screen below shows the input mask for a weekly program with the function Step:



The screen below shows the input mask for weekly program with the function Temperature.



The following main functions are available:

Cancel	Cancel the changes and go back to the Event Overview.
OK	Save the changes and go back to the Event Overview.
Choice	Shows the Select List of objects matching the Selection Criterion. For selection, double-click an item or click and press OK.
Insert	A list of available entries is on the right side of the button. The entry marked in this list is insert to the list on the left. The list shows the current Special Events or Commands for the actual object. When clicking an item in the list of available entries, the selected item in the list of Special Events or Commands is overwritten.
Del	Delete the selected item in the list of Special Events or Commands.
NOT	The selected Special Event can be negated (inverted) by clicking the NOT column of the entry. A second click cancels the negation.

The cursor can be set into a field by mouse click or by stepping through with the Tab key. Input fields are displayed with a white background.

The special **?** command allows you to enter a user specified value. It is indicated by a prompt ?= in the list of commands and accepts any useful value. Just enter the value after the prompt e.g. ?=23 or ?=Alarm.

You can limit an event to a Day of Week by simply checking the appropriate box and/or you can limit it to a Fix Date and/or Valid from Through. Note that periods for Day of Week, Special Events, can exclude each other and thus prevent an action. As a logical rule consider an Event to be valid at the time given by: [Day of Week **OR** Special Event] **AND** [Fix Date] **AND** [Valid from..Through] **AND** [Event Time].

An empty field generally means *always*. An empty time field means at *00:00:00*.

Selection Criterion

If you press Selection, a mask displays where the criterion for filtering can be entered. Note, a blank field will be interpreted as *all*. The syntax of a criterion depends on the field type where it is entered. More information about filtering is in the appendix *Regular Expressions* of the *SuperNova* user manual.

Overview of criterion in numerical fields (##=number):

##	field value must be equal ##
<##	field value must be smaller than ##
<=##	field value must be equal or smaller than ##
>##	field value must be greater than ##
>=##	field value must be equal or greater than ##
!=##	field value must be different (not equal) from ##

Overview of criterion in alphanumerical fields where even wildcards can be used:

*****	field must be empty (blank)
!*	field must have an entry (not empty)
***string**	field value must be equal to string
!*string	field value must be different from string
***^string**	field value must begin with string

Example: To see only the objects whose name begins with MB01, type ***^MB01** in the object name field.

Advanced selection criterion:

[A-Z]	one capital letter
[a-f]	one small letter from a through f
[0-9]	one digit
[A-Z]\{10}	10 consecutive capital letters
B[0-9]\{4\}	a B followed by four digits
[^0-9]*	no digits are allowed (^ in the bracket means NOT)
[A-Z]\{1,3\}[0-9]*	1, 2, or 3 capital letter followed by digits

PROGRAM ARGUMENTS

You can control the behavior of ETM by program arguments. A program argument is marked by a hyphen {-} followed by a argument name and a value if required. Program arguments are not case-sensitive and must be separated by at least one space. An argument without a hyphen is interpreted as a file name where the program arguments are read.

The ETM task writes information about startup, shutdown, version, actual program argument values and log output into file {flapp}\{FLNAME}\{FLDOMAIN}\{FLUSER}\log\etm.log. As an example, a log file name can be c:\flapp\flapp1\shared\shareusr\log\etm.log.

Argument	Description (also see sample file ETM_para.run on the installation media)	Default
`file`	**Program Argument File** The file must be specified by full path and file name in the System Configuration table. Environment variables can be used; they must be set in braces { }, such as {flapp}\etm_para.run.	None
`-OpMode#`	**Startup Command Operation Mode #=0..2** ETM will start up in the specified mode.	0
`-SleepTime#`	**Cyclic Sleep Time #=100..1000 [ms] (Normal Interval)** After each process cycle, ETM is suspended for the specified time if the internal clock is not more than 1 second behind the external clock (system clock or test clock). See ‹ Parameters <ClockTick>, <InternalTime> and <ExternalTime>.	900 ms
`-ShortSleepTime#`	**Cyclic Short Sleep Time #=50..500 [ms] (Catch-up Interval)** After each process cycle, ETM is suspended for the specified time if the internal clock is more than 1 second behind the external clock (system clock or test clock). See ‹ Parameters <ClockTick>, <InternalTime> and <ExternalTime>.	300 ms
`-StartupTime#`	**Span Considering Startup Events #=1..100 [hours]** At startup, ETM considers events in this span if the appropriate command uses Mode S to initialize the control tags value, see Function Table Mode column.	6 hours
`-BufferTime#`	**Additional Span for Events #=1..10 [days]** When scanning the Event List, ETM stores every event for the specified day(s). This span is used to prevent the loss of events in case the Event List is not available for any reason.	1 day
`-ReadPeriod#`	**Rate of Reading the Event List #=600..100000 [sec]** This is the period the Event List is scanned for events valid for this day and the time buffer. Use ‹ Parameter Tag <ReadDB> to force reading the Event List.	900 sec

Argument	Description (also see sample file ETM_para.run on the installation media)	Default
-MaxMsgSize#	**Maximum Length of String in MESSAGE Tags #=1..30000 [bytes]** This is the largest message that ETM will handle.	8096 bytes
-LogEvent0	**Event Log Type 0** Information of every processed event is written into the ETM log file (other types not supported)	None
-LogCtLoad	**CT-File Load Log** At startup of ETM, the results of reading the configuration tables and generating the input triggers are written to the ETM log file.	None
-LogTrigger	**Input Trigger Log** If an input trigger occurs at run time, it is written in the ETM log file.	None
-LogStack	**Function Calls Log** Every internal function call of ETM is written in the ETM log file.	None
-LogMemory	**Memory Management Log** Allocating and freeing memory are written in the ETM log file.	None
-LogStatus	**Status Messages Log** On the run-time manager, only the latest message of a task is visible. If this argument is set, every status message of ETM is written into the ETM log file for protocol information.	None

Program Arguments for Input Masks

When starting the ETM Input Masks, you can preselect the list of objects to display by the program argument OBJECTNAME=<expr>, where expr is a regular expression defining the filter.

Do not use an asterisk at the begin of **expr** although it is required in the selection mask. This can be useful in conjunction with MMI in order to display the Input Mask for a currently selected object; for example, to see the events of object named LK06MM01_AS, enter:
nova -w etm OBJECTNAME=LK06MM01_AS
or for events of the first object beginning with LK06, enter:**nova -w etm OBJECTNAME=^LK06**

INFORMATION AND ERROR MESSAGES

The following messages can display on the Run-time Manager. Note, %s, %d, etc. represent values created at run time, #nnn is a 3 digit unique message number, and %03d:%05d indicates a tag index.

Label in file ETM.TXT	Information and Warning Messages = #0xx
PROG_STARTING	ETM Vx.xx starting
PROG_RUNNING	ETM Vx.xx running
PROG_STOPPING	ETM Vx.xx stopping
PROG_INACTIVE	ETM Vx.xx normal shut down
PROG_FILE_PROC	Processing file %s
PROG_REC_DAT_PROC	Processing all records for internal use
PROG_NO_JOB	#001 No objects defined to schedule
ACE_INV_AUTHORIZATION	#002 No authorization: shut down by time-out
ACE_DRV_NOT_INSTALLED	#003 Driver not installed: shut down by time-out
ACE_PRINTER_NOT_READY	#004 Printer not ready: shut down by time-out
ACE_RESTART_PREVENTED	#005 Restart prevented: shut down FactoryLink first
FLOAT_VAL_WRONG	#010 Invalid float in assembled value
E_FLCHANGEREAD	#020 Error while FL_Change_Read code=%d
E_FLCLEARCHANGEFLAG	#021 Error clearing change flag tag %03d:%05d code=%d
E_FLREAD	#022 Error reading tag %03d:%05d code=%d
E_FLSETCHANGEFLAG	#023 Error setting change flag tag %03d:%05d code=%d
E_FLWRITE	#024 Error writing tag %03d:%05d code=%d
E_UNKNOWN_COMMAND	#040 Error command %s not allowed on function %s for object control tag %s
E_UNKNOWN_TRIGGER_TYPE	#041 Error unknown trigger type %d
E_UNKNOWN_VAL_TYPE	#042 Error unknown value type in function %s
E_UNKNOWN_SPECEVENT	#043 Error unknown Special Event %s defined on an event for object control tag %s

Label in file ETM.TXT	Information and Warning Messages = #0xx
E_EXTTIMEINPUT	#044 Error invalid time format in external time tag, use ISO 8601 (YYYY-MM-DD hh:mm:ss)
E_AF_GENERAL	#070 Error Event List ascii file general failure
E_AF_READ_RECORD	#071 Error reading Event List ASCII file record
E_AF_FILE_OPEN	#072 Error opening Event List ASCII file
E_AF_FILE_READ	#073 Error reading Event List ASCII file
E_AF_UNKNOWN_INTERFACE	#074 Error unknown interface for Event List ascii file
E_AF_WHERECLAUSE	#075 Error creating where clause for Event List ascii file
E_DB_GENERAL	#080 Error Event List general database error
E_DB_READ_RECORD	#081 Error reading Event List record
E_DB_READ_MOVEABS	#082 Error reading Event List position trigger
E_DB_READ_STATUS	#083 Error reading Event List status tag
E_DB_READ_CURRENTROW	#084 Error reading Event List current row tag
E_DB_WRITE_MOVEREL	#085 Error writing Event List move trigger
E_DB_WRITE_SELECT	#086 Error writing Event List select trigger
E_DB_WRITE_SQL	#087 Error writing Event List SQL statement

Label in file ETM.TXT	Failure Shutdown Messages = #1xx
SOFTWARE_NOT_LICENSED	#100 Option not installed or License not enabled Verify the option key is installed and the license is enabled. Use the License Wizard to see the purchased options.
ACE_ERR_IN_INSTALLATION	#101 Authorization executable file may be corrupted
ACE_ERR_IN_KEYFILE	#102 Authorization key file may be corrupted
PROG_INIT_FAIL	#103 Error registering ETM to kernel, code=%d
E_CT_GET_HDRLEN	#110 Error header length %s in ct file %s len=%d
E_CT_GET_NCTS	#111 Error empty ct file %s
E_CT_GET_NRECS	#112 Error no records in ct file %s
E_CT_GET_RECLEN	#113 Error record length %s in ct file %s len=%d
E_CT_OPEN	#114 Error opening ct file %s
E_CT_READ_HDR	#115 Error reading header %s in ct file %s
E_CT_READ_INDEX	#116 Error reading index in ct file %s
E_CT_READ_RECS	#117 Error reading %s records in ct file %s
E_CT_TYPE	#118 Error unknown ct type %d in ct file%s
SYS_NO_MEMORY	#130 Error getting memory
SYS_FOPEN_ERR	#131 Error opening file %s
PROG_THREAD_START_ERR	#132 Error starting thread %s : %s
E_GET_GLOBAL_TAG	#133 Error global tag with id=%d not found
E_DCREATE	#134 Error creating directory %s
E_MKDIR	#135 Error creating directory %s
E_NPATH	#136 Error getting memory for NPATH

Label in file ETM.TXT	Configuration Error Shutdown Messages = #2xx
PROG_ARG_UNKNOWN	#201 Program argument %s unknown
E_PROGENVVAR	#202 Environment variable %s unknown
E_FOPENARG	#203 Error opening program argument file %s
E_WRONG_TAGTYPE	#210 Error wrong tag type %d for tag %s
E_UNKNOWN_LISTFORMAT	#220 Error unknown format %s of Event List
E_DB_ALIAS_NOT_FOUND	#221 Error database alias %s not found in historian table which handles the Event List
E_DB_ALIAS_NOT_LOCATABLE	#222 Error database alias not locatable in DBBROWSER table “%s”
E_DB_FIELD_NOT_FOUND	#223 Error field %s not found in DBBROWSER table %s
E_LOGEXPR_COUNT	#224 Error count of logical expressions lines not %d
E_LOGEXPRTAG_NOT_FOUND	#225 Error tag describing logical expression „%s“ not found in DBBROWSER table %s
E_EVENTLIST	#226 Error Event List database schema
E_DUPOBJECT	#227 Error duplicated object control tag %s
E_DUPFUNCTIONANDCOMMAND	#228 Error duplicated command %s on function %s
E_NOFUNCTION	#229 Error function %s for object control tag %s not defined
E_INVALID_PARAARG	#230 Error invalid Parameter argument %s, no white-space characters allowed

Chapter 11 . . . .

File Manager

File Manager allows you to perform basic operating system file management operations initiated by a FactoryLink application at run time. This task works in conjunction with FactoryLink's FLLAN option to initiate operations within other FactoryLink stations on the network.

Operating Principles

The File Manager initiates the following operations:

- Copy a file
- Rename a file
- Delete a file
- Print a file
- Display a directory
- Type a file

File Manager initiates these operations with six commands:

- COPY	- PRINT
- REN	- DIR
- DEL	- TYPE

These commands perform the same functions just like their operating system counterparts. The File Manager controls all file operations through the FactoryLink real-time database.

You can configure other FactoryLink tasks to initiate File Manager operations. For example:

- You can configure input functions in Graphics so an operator can use them to initiate file-management operations at run time, such as to display a list of recipes or reports.
- The Timer task can trigger File Manager to automatically back up files to a networked server at certain intervals, such as each day at midnight.
- The Timer task can also trigger File Manager to delete log files automatically at certain intervals, such as once every four hours, or after certain events, such as when log files reach a specified size.
- Alarm Supervisor can trigger File Manager to print alarm files.

To enable a local node to perform file management operations with a remote node at run time, the FLFM_SERVER task must be running on the node that does not initiate the FLFM command. Either start the FLFM_SERVER task manually from the Run-Time Manager screen at startup or configure FactoryLink to start the FLFM_SERVER task automatically at system startup. Complete the following steps to do this.

1. Open the System Configuration Information table on the remote node.
2. Locate the row containing the entry FLFM_SERVER in the **Task** field.
3. In the **Flags** field for that row, enter an **R**. This configures the FLFM_SERVER task on the remote node to start up automatically whenever FactoryLink is started.

FILE MANAGER CONTROL TABLE

The File Manager Control table sets up triggers for the File Manager commands you want to perform and required source and destination path information.

File manager defaults to the User domain, but you can configure it in to run in either the Shared or the User domain.

Accessing

In your server application, open **Other Tasks > File Manager > File Manager Control.**

Field Descriptions

Table Name Specifies the name of the operation being defined or modified. For TYPE and DIR operations, this field connects the entry in the File Manager Control table with the associated File Manager Information table for that entry.

This field is optional for COPY, REN, DEL, and PRINT operations and is required for TYPE and DIR operations. You can use it to distinguish different operations of the same type.

Valid Entry: up to 16 alphanumeric characters

Command Trigger Name of a tag used to initiate the file operation.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

Position Trigger Name of a tag that tells the File Manager where in a directory to start listing files or where in a file to start typing. The File Manager starts reading after the line number specified by the value of the Position Trigger tag.

For example, if the value of the Position Trigger tag is 6, the File Manager begins reading the file at line seven. The number of lines displayed or the number of files listed depends on the number of tags defined in the File Manager Information table. You can configure the system so any FactoryLink task can change the value of this tag at run time so the File Manager starts at a different point in a directory or file.

Required only for DIR and TYPE operations; not used for COPY, PRINT, REN, and DEL operations. Do not specify the same tag name for DIR and TYPE operations.

Valid Entry: tag name
Valid Data Type: analog

Command — File operation to be performed. This can be one of the following:

COPY Copies the source file to the destination file. Does not require a File Manager Information table. Only one file in a copy operation can be remote.

REN Renames the source file to the destination file. Both the source and destination paths must be the same. Does not require a File Manager Information table.

DEL Deletes the source file. The DEL command requires only the source path; the destination path is ignored. Does not require a File Manager Information table.

PRINT Causes the file specified by the source path to be printed on the device specified by the destination path. The destination path must contain the name of a device known to the Print Spooler task. This operation works only with the Print Spooler and does not require a File Manager Information table.

DIR Displays a list of all files in the directory specified by the source path, which can include wildcard characters. The destination path is ignored. This operation requires you to complete a File Manager Information table and to animate Text objects in mimics to display lines from the file.

TYPE Displays the contents of the source file. The destination path is ignored. This operation requires you to complete a File Manager Information table and to animate Text objects in mimics to display lines from the file.

Source File Spec.

Full path name of the source file. The **source file spec.** can use the file name syntax for the operating system of either the remote or the local station.

If you are using FLLAN, the source can reference a remote station. The two stations must be the same if you specify a remote station for the destination and a remote station for the source. You have no restriction on the source station if the destination is local or on the destination if the source is local. For more information about referencing remote stations, see “Using File Manager with Networks” on page 217.

Type wildcard characters in the path name to show a root directory’s contents using the **DIR** command. For example, in this field type:

C:*.*

For standalone systems:

/DEVICE_NAME/DIR_NAME/SUB_DIR_NAME/FILE_NAME

For networked systems:

\\STATION_NAME\DEVICE_NAME/DIR_NAME/SUB_DIR_NAME/FILE_NAME

In the case of the COPY, DEL, and DIR commands, the source file specification may contain variable specifiers or wildcard characters (*). For more information about variable specifiers and wildcard characters, see “Using Variable Specifiers in File Specifications” on page 214 and “Using Wildcard Characters in File Specifications” on page 216.

Valid Entry: full path name

Note: When using environment variables in path names, you can enter the name of the environment variable surrounded by braces { } and FactoryLink extends the pathname using the default setting. For example, use **{HOME}/flink/csinfo.txt** for DIR and TYPE.

Source Variables 1-4

Names of tags whose values replace the variable specifiers in the source path name. These fields work in conjunction with the **Source File Spec.** field to form the path of the file the File Manager performs operations to. The value of the tag in the **Source Variable 1** field replaces the first variable specifier, the value of the tag in the **Source Variable 2** field replaces the second variable specifier, and so on.

If the tag specified in this field is undefined, the Tag Editor appears when you click **Enter**.

Ensure the data type of the tag matches the variable specifier type if using variable specifiers.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

Destination File Spec.

Full path name of the destination file. The **Destination File Spec.** can use the file name syntax for the operating system either the remote or the local station resides on.

If you are using FLLAN, the destination can reference a remote station. The two stations must be the same if you specify a remote station for the destination and a remote station for the source. You have no restriction on the source station if the destination is local or on the destination if the source is local. For more information about referencing remote stations, see “Using File Manager with Networks” on page 217.

For standalone systems:

/DEVICE_NAME/DIR_NAME/SUB_DIR_NAME/FILE_NAME

For networked systems:

\\STATION_NAME\DEVICE_NAME/DIR_NAME/SUB_DIR_NAME/FILE_NAME

Unless you use wildcard characters in the source file specification, specify the full path name of the destination. If you use wildcard characters, do not specify the full path; specify only the directory.

In the case of the COPY, DEL, and DIR commands, the destination file specification can contain variable specifiers or wildcard characters (*).

Use the following destination file specification format if using the PRINT command:

[\\station_name\] [flags] [spool_device]

where

station_name Is the optional FactoryLink station name (defaults to LOCAL). Not used on standalone systems.

flags Optional flags, which can be one of the following.

NONE—This is the default.

B—Binary file.

S—Suppress Beginning and End of File. Used to concatenate files.

spool_device Is the optional spool device (defaults to 1; legal devices are 1 through 5).

Valid Entry Format	Description	Example
DRIVE:\DIR\SUBDIR\ FILE.EXT	For standalone systems	C:\LOG\HISTORY\ HISTORY.ARC
\\STATION\DRIVE:\ DIR\SUBDIR\FILE.EXT	For networked systems	\\NODE1\C:\HIST\PIC\ SCREEN1.PIC

Destination Variables 1-4

Names of tags whose values replace the variable specifiers in the destination path name. These fields work in conjunction with the **Destination Format** field to form the path of the file the File Manager performs operations in. The value of the tag in the **Destination Variable 1** field replaces the first variable specifier, the value of the tag in the **Destination Variable 2** field replaces the second variable specifier, and so on.

Ensure the data type matches the variable specifier type if you use variable specifiers.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

Completion Trigger

Name of a tag used to indicate a file-management operation is complete, but not necessarily successful. This tag, if defined, is set by File Manager and can be referenced by any FactoryLink task, including File Manager to monitor file-management operations or trigger an event.

Valid Entry: tag name
Valid Data Type: digital

Completion Status

Name of a tag set by the File Manager task to indicate the status of an operation. The Completion Status tag can be referenced by any FactoryLink task, including the File Manager to handle file error situations or trigger the next File Manager table to start an operation.

Valid Entry: tag name
Valid Data Type: analog

The File Manager writes an analog value to the Completion Status tag to indicate the status of a file management operation. In addition to the Completion Status tag, you can use the task's TASKMESSAGE_U[x] tag to report messages on the application screen.

This tag can have any of the following status values.

Value	Description	Value	Description
0	Operation completed successfully.	12	Can't open file (path/file name).
1	Current operation in progress.	13	Error occurred while reading a file.
2	Specified file(s) not found.	14	File could not be created.
3*	Requested a line beyond end of file.	15	Error occurred while writing to a file.
5	Remote system could not perform requested action.	16	Illegal spool device was specified.
6	Attempt to log onto a remote station failed	17	Not enough memory to perform operation
8	Network transmission error occurred	97	Illegal file name was specified
* If the TYPE/DIR position trigger offset values are increased beyond the end of the file, the File Manager reads in as many lines as possible, sets the completion trigger, and sets the completion status to 3.			

For examples of File Manager operations, see "Sample File Manager Operations" on page 208.

FILE MANAGER INFORMATION TABLE

The File Manager Information table defines the tags that will display the results of a TYPE or DIR operation. Animate these tags as Text objects on a graphics screen. Complete this table ONLY if you are configuring a DIR or TYPE operation. Otherwise, you have completed the configuration of the File Manager task.

Accessing

In your server application, open **Other Tasks > File Manager > File Manager Control > "your tag name" > File Manager Information.**

Field Descriptions

Tag Name — Name of a message tag that, as a result of a DIR or TYPE command, receives a message value to be displayed in a single line on a graphics screen. The number of **Tag Name** fields defined in this table determines the number of lines displayed as a result of a DIR or TYPE command at run time. (Required only for DIR and TYPE operations; not used with COPY, PRINT, REN, and DEL operations.)

If the tag specified in this field is undefined, the Tag Editor appears when you click **Enter** with a tag type of message in the **Type** field. Accept this default.

Values are written to the tags defined in the **Tag Name** field whenever a DIR or TYPE operation is triggered or whenever the operator changes the value of the Position Trigger tag defined in the control table. A different value in the Position Trigger tag means information from a different place in the directory or file is displayed.

Valid Entry: tag name
Valid Data Type: message

SAMPLE FILE MANAGER OPERATIONS

The examples below illustrate each type of file-management operation. In these examples, entries for each type of file-management operation are displayed in separate sample Control tables. In an actual FactoryLink application, all file-management operations are configured in one control table.

The first four examples do not require you to complete an associated File Manager Information table. The last two examples do require you to complete a File Manager Information table.

Example 1: COPY

Example 1 demonstrates how to configure a COPY operation using Windows file syntax. You can configure the Math & Logic task or an analog counter in the Counters task to use this operation to increment the alarm history file number. This results in a rolling count of the history file being transferred: Hist.001, Hist.002, and so on. Complete the control table to configure a COPY operation,

Sample File Manager Control Table		
Field	**Sample Entry**	**Explanation**
Table Name	WINCOPY	Leave this field blank because the COPY operation does not require completion of the File Manager Information table; however, we recommend you complete this field to distinguish different operations of the same type.
Command Trigger	copytrig	Name of the digital tag that triggers the copy operation. You can configure other tasks to write to this tag to trigger the copy operation.

Sample File Manager Control Table		
Field	**Sample Entry**	**Explanation**
Command	COPY	Designates a COPY operation.
Source File Spec.	c:\history\ hist.%03d	The path and file name of the file to be copied. In this example, the source path name contains a variable specifier. At run time, whenever the operator or another task triggers the digital tag **copytrig**, File Manager replaces this variable with the value of the analog tag **fx1_ext**.
Source Variable 1	fx1_ext	At run time, File Manager replaces the variable %03d in the **Source File Spec.** field with the value of this analog tag. In this example, this value is a three-number file extension, such as .001.
Destination File Spec.	a:\archive	The path name where the source file is to be copied. Destination Variables are not necessary because this entry contains a static specifier.
Completion Trigger	copydone	When the COPY operation has been completed, File Manager forces the value of this tag to 1 (ON), indicating the operation is complete.

Completion Status	copystat	When the COPY operation is complete, File Manager indicates the status of the operation by writing an analog value to this tag. Use the task's TASKMESSAGE_U[6] tag to report messages on the application screen in addition to the Completion Status tag. For example, if you get a status 12 in the Completion Status tag, you also see the message: Can't open file (path/file name).
		
		If you want to use Math and Logic for this function, set up a math procedure triggered by the status tag itself. In the procedure, assign the appropriate ASCII string for each possible status code to a message tag in a series of IF THEN statements. For example,
		
		IF status = 1 THEN
		
		MSG = "whatever"
		
		ENDIF

Example 2: PRINT

Example 2 demonstrates how to configure a PRINT operation. Configure the control table to configure a PRINT operation. PRINT command file syntax is the same for all operating systems.

Sample File Manager Control Table		
Field	**Sample Entry**	**Explanation**
Table Name	PRINT	Designates the name of a PRINT operation.
Command Trigger	printtrig	Name of the digital tag that triggers the print operation. You can also configure other tasks to write to this tag to trigger a print operation.
Command	PRINT	Designates a PRINT operation.
Source File Spec.	%s	The name of the file to be printed. In this example, the file name is a variable specifier. At run time, File Manager replaces this variable with the value of the message tag **printpath**, specified in the **Source Variable 1** field.
Source Variable 1	printpath	At run time, the File Manager replaces the variable %s in the **Source File Spec.** field with the value of this message tag. In this example, this value is a file name.
DestinationFile Spec.	1	1 designates the print spool device that prints the file specified by **printpath**. This example assumes you have already defined print spool device 1 in the Print Spooler task. At run time, when the operator or a task triggers the digital tag **printtrig**, File Manager prints the file to Print Spooler device number one.

Example 3: REN (Rename)

Example 3 demonstrates how to configure a REN (rename) operation using Windows file syntax. Configure the control table to configure a REN operation.

Sample File Manager Control Table		
Field	**Sample Entry**	**Explanation**
Table Name	WINREN	Designates the name of a REN operation.
Command Trigger	rentrig	Name of the digital tag that triggers the rename operation.
Command	REN	Designates a REN operation.
Source File Spec.	c:\temp\%s.log	This field contains a variable specifier, which File Manager replaces at run time with the value of the message tag **s_file**, specified in the **Source Variable 1** field.
Source Variable 1	s_file	File Manager replaces the variable %s in the **Source File Spec.** field at run time with the value of this analog tag.
Destination File Spec.	%s.tmp	**%s.tmp** contains a variable specifier, which File Manager replaces at run time with the value of the message tag **rento**, specified in the **Destination Variable 1** field.
DestinationVariable 1	rento	When the value of the digital tag **rentrig** is forced to 1 (ON), File Manager renames the file specified in **s_file** with the file name contained in **rento**.

Example 4: TYPE

Example 4 demonstrates the TYPE command. TYPE command file syntax is the same for all operating systems.

Sample File Manager Control Table		
Field	**Sample Entry**	**Explanation**
Table Name	TYPE	Designates the name of a TYPE operation.
Command Trigger	typetrig	Name of the digital tag used to trigger the type operation.
Command	TYPE	Designates a TYPE operation.
Position Trigger	typescroll	Name of the analog tag that controls the output of the TYPE command
Source File Spec.	%s	This field contains a variable specifier, which the File Manager replaces at run time with the value of the message tag **typepath**, designated in the **Source Variable 1** field.
Source Variable 1	typepath	At run time, when the value of the digital tag typetrig is forced to 1 (ON), the File Manager types the file specified by **typepath**. Because TYPE operations do not need a destination, the **Destination Variable** fields are left blank.
Tag Name	typlin1 typlin2 typlin3 typlin4 typlin5 typlin6 typlin7 typlin8	At run time, when the value of the digital tag **typetrig** is forced to 1 (ON), the File Manager reads the file specified in the Source Tag 1 tag **typepath**. The File Manager starts reading after the line number specified by the Scroll tag **typescroll**. In this example, eight lines of text from the specified file are read into message tags **typlin1**, **typlin2**, ...**typlin8**. These message tags may be referenced by other FactoryLink tasks or displayed on a graphics screen.

Example 5: DIR (Directory)

Example 5 demonstrates the DIR (Directory) command. DIR command file syntax is the same for all operating systems.

Sample File Manager Control Table		
Field	**Sample Entry**	**Explanation**
Table Name	DIR	Designates the name of a DIR operation.
Command Trigger	dirtrig	Name of the digital tag used to trigger the **dir** operation.
Position Trigger	dirscroll	Name of the analog tag that controls the output of the DIR command.
Command	DIR	Designates a DIR operation.
Source File Spec.	%s	This field contains a variable specifier, which the File Manager replaces at run time with the value of the message tag dirpath, designated in the **Source Variable 1** field.
Source Variable 1	dirpath	At run time, when the value of the digital tag dirtrig is forced to 1 (ON), File Manager displays the directory specified by **dirpath**. The **Destination Variable** fields are blank because DIR operations do not need a destination.
		At run time, when the value of the digital tag **dirtrig** is forced to 1 (ON), File Manager reads the directory specified in the Source Tag 1 tag **dirpath**. File Manager starts reading after the line number specified by the Scroll tag **dirscroll**. In this example, eight lines of text from the specified file are read into message tags **dirlin1**, **dirlin2**, ...**dirlin8**. These message tags may be referenced by other FactoryLink tasks or displayed on a graphics screen.

Example 6: DEL (Delete)

Example 6 demonstrates the DEL (Delete) command using Windows file syntax.

Sample File Manager Control Table		
Field	**Sample Entry**	**Explanation**
Table Name	WINDEL	Designates the name of a DEL operation.
Command Trigger	deltrig	Name of the digital tag used to trigger the delete operation.
Command	DEL	Designates a DEL operation.
Source File Spec.	c:\hist\%s.tmp	This field contains a variable specifier, which File Manager replaces at run time with the value of the message tag **delfile**, designated in the **Source Variable 1** field.
Source Variable 1	delfile	At run time, when the value of the digital tag **deltrig** is forced to 1 (ON), File Manager deletes the file specified by **delfile**. The **Destination Variable** fields are left blank because DEL operations do not need a destination.

USING VARIABLE SPECIFIERS IN FILE SPECIFICATIONS

Use variable specifiers if the operator or the system is to enter all or part of the source or destination specification; however, do not use variable specifiers if the same files are to be used for all operations. Use the **Source and Destination Variable** fields to designate variables to replace the variable specifiers.

You can include up to four variable specifiers (each one designated by a leading percent sign %) in the path or file name. These variable specifiers indicate a portion of the path or file name that is variable (replaced with data from tags when the file operation is performed). The variables can be digital, analog, long analog, floating-point, or message tags. Multiple variables can be used together, as in a file name and extension (for example, %8s.%3s).

If you want to vary the actual path/files used in either the source or destination paths, use one or more of the four variables and %xx type specifiers to dynamically build these at run time from tags; otherwise, hardcode the exact path/file names desired and leave the four tag variable fields blank.

The data type of the tag must match the variable-specifier type as follows.

Variable Specifier	Variable Specifier Type	FactoryLink Data Type
%d	Decimal	Digital, Analog, Longana, Float
%s	String	Message

The table below contains examples of variable specifiers using generic syntax.

Variable Specifier Examples		
Variable Description	**Entry**	**Sample Results**
Directory, subdirectory, or file name	%8s	FLINK
Numeric file extension	.%d	.1, .2, .3
	%.03d	.001, .002, .003
File name and extension	%8s.%03d	file.001
	%8s.%3s	file.exe

The following table contains Windows-specific examples of variable specifiers used to designate paths. The last example shows the path name of a station on a network.

Samples of Variable Specifiers in Path Names		
Description	**Example**	**Comments**
Standard path	c:\history\exe\sample.exe	Base example
1 variable	c:\%8s\exe\sample.exe	Variable for directory
2 variables	c:\%8s\%8s\sample.exe	Variables for directory and subdirectory
3 variables	c:\%8s\%8s\sample.%3s	Variables for directory, subdirectory, and 3-character file extension
4 variables	%1s:\%.8s\%.8s\ sample.%3s	Variables for drive, directory, subdirectory, and file extension
Standard path for a network station	\\node2\c:\history\exe\sample.exe	\\node2\ is the name of a remote station

USING WILDCARD CHARACTERS IN FILE SPECIFICATIONS

The File Manager task accepts the following wildcard character specifiers in the source and destination path names:

* (asterisk)	For string replacement.
? (question mark)	For single-character replacement.

Format path names:

source	**/DEVICE_NAME/DIR_NAME/SUB_DIR_NAME/WILDCARD FILESPEC.**
destination	**/DEVICE_NAME/DIR_NAME/SUB_DIR_NAME.** Do not specify a file name for the destination path with COPY as File Manager does it for you.

Path names with wildcard characters in the file specifications might resemble this example:

source	**/DEVICE/FLINK/SAMPLE/SAMPLE.***
destination	**/DEVICE/FLINK/EXE.**

Example of a File Manager operation using wildcard characters Windows file syntax):

Sample File Manager Control Table Using Wildcard Characters		
Field	**Sample Entry**	**Explanation**
Command	COPY	Designates a COPY operation.
Trigger	copytrig	Name of the digital tag used to trigger the copy operation.
Source Format	c:\hist\report\rpt*.*	Designates the source path.
Destination Format	C:\HIST\ RECORDS	At run time, all files in the REPORT subdirectory that begin with RPT are copied to the RECORDS subdirectory.

Do not specify a file name for the destination path as File Manager will do it for you. Pathnames with wildcard characters in the file specifications might resemble this example:

source	**C:\FLINK\SAMPLE\SAMPLE.***
destination	**C:\FLINK\EXE**

USING FILE MANAGER WITH NETWORKS

In addition to being fully compatible with FactoryLink, the FactoryLink File Manager is compatible with networks using the TCP/IP protocol type.

File-management functions, such as copying, deleting, printing, and renaming files, can be performed between the local FactoryLink system and any remote computer running File Manager as long as the FactoryLink system contains the FactoryLink Local Area Networking (FLLAN) option.

If using FLLAN, create the LOCAL file before filling in the configuration tables. Define the local station name in the ASCII file LOCAL in the **FLAPP/NET** directory. Remember: standalone systems require the LOCAL file.

Either the source or destination path name can refer to a file on a remote station. The format for a remote file path is

\\(station)\(path)

where

station	Is the name of the remote station, up to 256 characters.
path	Is the full path name of the file on the remote station.

The source and destination are interchangeable as long as one of them is the local FactoryLink station. The only difference between local file operations and remote file operations is remote file names must include the disk/drive specification if required by the operating system and must conform to the file name syntax for the remote computer's operating system.

Only one file can be remote in a copy operation. Both files must be on the same station in a rename operation.

For example, to copy a file from a local FactoryLink station to a remote FactoryLink station, use the following format for the remote path name:

\\STATION_NAME\DEVICE_NAME/DIR_NAME/FILE_NAME

Other file-management operations can be performed with File Manager using the same format.

Do not use the remote file name (\\(STATION)\) when performing File Manager operations on networks unless you installed FLLAN on the local and remote computers. Using the FLLAN FactoryLink station name instructs FLLAN rather than the network to perform the operation.

At run time, ensure the FLFM_SERVER task is running on the remote node before invoking file management operations between local and remote nodes.

Using the COPY Command with FLLAN

COPY operations are the most common File Manager operations performed on a networked system and have the following uses:

- Copy local files to a remote station
- Copy remote files to a local station
- Copy local files to the local station

Example of a copy operation on a networked system (Windows file syntax)

Sample File Manager Control Table		
Field	**Sample Entry**	**Explanation**
Table Name	WINXFER	Designates a file-transfer operation on a network using FLLAN.
Command	COPY	Designates a COPY operation.
Trigger	filexfertrig	Name of the digital tag used to trigger the copy operation.
Source Format	c:\arc\ %s.log	Source path where **%s** is the variable specifier. The string specified by the value of the message tag **s_file** replaces the %s in the source specification.
Source Tag 1	s_file	At run time, File Manager replaces the variable **%s** in the Source Format field with the value of this analog tag.
Destination Format	\\nod2\%s	File Manager computes the destination path in the same way as the source path.
Destination Tag 1	s_path	If the message tag **s_path** contains the following message: **{FLINK}\RECIPE** the computed destination specification is **\\NOD2\{FLINK}\RECIPE**
Completion Trigger	copydone	File Manager forces the value of this tag to 1 (ON) when the COPY operation is complete. Another FactoryLink task can use this tag or to trigger another File Manager operation.
Completion Status	copystat	File Manager writes the status information to this analog tag when the COPY operation is complete.

For information about variable specifiers, see the *Fundamentals Guide*.

Using the COPY Command with a Network Without FLLAN

When specifying a COPY command for networks without FLLAN, do not specify a station name in the Source Format and Destination Format fields. Instead, specify a network device that is any logical device on a remote computer used as if it were a local device.

Different operating systems reference network devices in different ways. Consult the user's manual for the appropriate operating system to find the proper syntax for referencing these devices.

PROGRAM ARGUMENTS

Task	Argument	Description
FLCM	-a<x> Where <x> is the ACK timeout in seconds	Each request is acknowledged. File Manager will wait a default of 15 seconds for the ACK. Use this parameter if you wish to change this time-out.
FLFM	-C	Do not replace first period in file name with a space.
	-D<#>	Set verbose level. (# = 0 to 22)
	-L	Enables logging of debug information to a log file.
	-S<#>	Closes and reopens log file every # messages.
	-T	Insert timestamp at beginning of each debug statement.
	-W<#>	Wraps log file every # messages.

ERROR MESSAGES

Error Message	Cause and Action
Can't determine fluser name	Cause: No **FLUSER** name is specified. Action: Specify an **FLUSER** name.
Can't find Network	Cause: Either the network shut down or is not installed correctly. Action: Restart the network if it shut down. Reinstall it if installed incorrectly. Cause: The wrong name is entered in the **LOCAL** file. Action: Enter the name the network is registered under in the **LOCAL** file. Cause: The ETC services file may not have any entries if using a TCP/IP network. Action: See “FLLAN” on page 227 for information about adding the correct entries to the ETC services file.
Can't add local name to the network	Cause: Either the name specified in the LAN Local Names table is not a valid network name or the network software may not have been installed properly. Action: In Configuration Explorer, open Local Area Network Groups and modify the local names on the LAN Local Names table. If error is not corrected, ensure the network software is operating properly.
Can't determine local node name	Cause: Either the LAN local names file (**FLAPP/NET/LOCAL**) does not exist or it contains invalid data. Action: Ensure **FLAPP/NET/LOCAL** exists. Create it if it does not exist. If it exists, find and correct any invalid data.
Can't open file *filename*	Cause: The specified file may not exist or another task may have opened it. Action: Ensure the specified file exists. Create it if it does not exist.
Can’t send ACK to inform client I'm ready	Cause: The client or network may not be operating. Action: Check state of client and network connection.

Error Message	Cause and Action
Error creating temp file; check SPOOL dir	Cause: The hard disk is full. Action: Delete any unnecessary files. Cause: The **/SPOOL** directory was not created during installation. Action: Create a **/SPOOL** directory or reinstall FactoryLink. Cause: The **/SPOOL** directory was deleted. Action: Create a **/SPOOL** directory or reinstall FactoryLink. Cause: The permissions are corrupt. Action: Change the permissions so the FactoryLink user name is authorized to create files in the **/SPOOL** directory.
Error opening CT	Cause: The **/FLINK/CT/FLFM.CT** file does not exist, may not be opened, or may be damaged. The installation may not have completed successfully. Action: Ensure the **/FLINK/CT/FLFM.CT** file exists. If the file does exist, delete **/FLINK/CT/FLFM.CT** and restart the application to rebuild the **FLFM.CT** file.
Error opening filename	Cause: The file does not exist. Action: Specify a different file or create the file. Cause: A syntax error occurred when specifying the path or filename. Action: Retry the command using the correct path and filename. Cause: The File Manager task does not have permission to read the specified file. Action: Change the permissions so the FactoryLink user name is authorized to open files in the specified directory.
Error printing file	Cause: Either the printer is disconnected, is not powered on, is not online, or has some other problem. Action: Ensure the printer cables are connected, the power is on, and the printer is online. If the file still does not print, contact the system administrator for the printer in use.
Error reading CT index	Cause: The **/FLINK/CT/FLFM.CT** file is damaged. Action: Delete **/FLINK/CT/FLFM.CT**. Restart the application to rebuild the **FLFM.CT** file.

Error Message	Cause and Action
File Manager task init. failed, err = *number*	Cause: The File Manager task could not register with the FactoryLink kernel. Action: Ensure the System Configuration Table contains the entry FLFM. Also, ensure the File Manager option is enabled on the FactoryLink key.
File Manager option bit not set	Cause: The option bit for the File Manager task is not enabled on the FactoryLink key. Either you entered the configuration sequence incorrectly during installation or you have not purchased the File Manager option. Action: First, check the license material on the Exhibit A form that accompanies the FactoryLink package to see if File Manager is listed as a licensed option. If it is listed on the Exhibit A form, reenter the configuration sequence and authorization code to enable the option. For information about the configuration sequence and authorization code, see the *Installation Guide*. If File Manager is not listed on the Exhibit A form, contact your FactoryLink sales representative to purchase the File Manager option.
File not found	Cause: The file does not exist. Action: Specify a different file. Cause: A syntax error occurred when specifying the path or filename. Action: Retry the command using the correct path and filename. Cause: The File Manager task does not have permission to print the specified file. Action: Change the permissions so the FactoryLink user name is authorized to print files in the specified directory.

Error Message	Cause and Action
File read ERROR on *filename*	Cause: The disk may be corrupt. Action: For Windows, run the disk diagnostic program CHKDSK to determine if the disk is corrupt. If the disk is corrupt, recreate the file from scratch or from the backup disk or tape. If the corruption is not on the root drive, inform your system administrator of the corruption as **fsck** will not be able to repair it.
File write ERROR on *filename*	Cause: The disk may be full. Action: Delete any unnecessary files. Add disk space if this error occurs frequently.
FLFM port in use-do file operation later	Cause: If the File Manager server gets this error, another server may already be up on that machine. If the File Manager client gets this error, then too many clients may be running on that machine. (The current limit is 10.) Action: Check for existing File Manager server or clients. Try the file operation again later.
FLFM port in use-do file operation later	Cause: If the File Manager server gets this error, another server may already be up on that machine. If the File Manager client gets this error, then too many clients may be running on that machine. (The current limit is 10.) Action: Check for existing File Manager server or clients. Try the file operation again later.
Internal request queue is corrupt	Cause: An internal FLFM error occurred. Action: Contact your technical support representative.
Invalid character in PRINT destination	Cause: An invalid character is entered after the node name in the **Destination Format** field of the File Manager Control table. Action: Open the File Manager Control table. Specify a valid entry in the **Destination Format** field.

Error Message	Cause and Action
Invalid command	Cause: An invalid command is entered in the **Command** field of the File Manager Control table. Action: Open the File Manager Control table. Specify a valid command in the **Command** field: **COPY** **PRINT** **DEL** **REN** **DIR** **TYPE**
Invalid CT file	Cause: The **/FLINK/CT/FLFM.CT** file is damaged. Action: Delete **/FLINK/CT/FLFM.CT**. Restart the application to rebuild the **FLFM.CT** file.
Invalid local name file	Cause: An invalid character or parameter is entered in the **LOCAL** field. Action: In your server application, open **Networking > Local Area Network Groups > local**. Ensure all information defined on the LAN Local Names table is valid. Correct invalid information. See “FLLAN” on page 227 for information about the correct syntax for this file.
Local wild-card copy failed on *filename*	Cause: Not enough RAM to build the file list. Action: Shut down unnecessary tasks. Add RAM if this error occurs often.
Network open did not connect	Cause: The session is not properly connected. Action: Retry the operation.
Network open failed *network name*	Cause: Either no available sessions exist or the node name is not valid. Action: Ensure the remote node name (part of the file path name) is valid and retry the operation.
Network open timeout	Cause: The network may be down. The node may be down. The node name in the file path name may be invalid. Action: Ensure the node name is valid and the node and network are up and running.

Error Message	Cause and Action
Network write error	Cause: The system cannot write to the network because the remote program shut down or terminated communications. Action: Ensure the remote node is up and connected.
Never got ACK from server-giving up	Cause: Server or network may be down or may have gone down. Action: Check state of server and network connection.
New client communications setup failed	Cause: The client or network may not be operating. Action: Check state of client and network connection.
No network for receive	Cause: You have specified a remote node name for a File Manager operation; however, the FactoryLink system does not contain the FLLAN option. File Manager operations cannot be performed to remote nodes unless the FLLAN option is installed. Action: Specify a local node or install the FLLAN option.
No network for send	Cause: You have specified a remote node name for a File Manager operation; however, the FactoryLink system does not contain the FLLAN option. File Manager operations cannot be performed to remote nodes unless the FLLAN option is installed. Action: Specify a local node or install the FLLAN option.
No tables configured for this task	Cause: No File Manager configuration tables have been defined. Action: In your server application, open **Other Tasks > File Manager > File Manager Control** and specify information in the File Manager's configuration tables.
Out of RAM	Cause: Not enough memory to run this task. Action: Restart the task. If it fails again, allocate more memory for this task or allocate more virtual memory.
Partial wild-card copy *filename*	Cause: An error occurred during a multi-file copy. The File Manager copied all files in the list prior to the specified file. The File Manager did not copy the specified file and any files after it on the list. Action: Copy the files that were not copied or retry the operation.

Error Message	Cause and Action
Received error or hangup during connect	Cause: The client or network may not be operating. Action: Check state of client and network connection.
Received no ACK or BUSY msg from server	Cause: Server or network may be down or may have gone down. Action: Check state of server and network connection.
Remote wild-card copy failed	Cause: Not enough RAM exists to build the file list. Action: Shut down unnecessary tasks. Add virtual memory if this error occurs often.
Request to allocate zero size RAM block	Cause: Either the operating system, third-party software, or hardware setup on your system is incorrect or incompatible or an internal error occurred. Action: Run a disk diagnostic program to determine if the disk is corrupt. If it is, recreate the files from a backup disk.
Server busy-waiting for ACK from server	Cause: The server is currently servicing another client's request. Action: No action is needed. The server will automatically serve the request when it is freed up.
Two different remote nodes, *node name node name*	Cause: A different remote name is entered in the **Source Format** field than was entered in the Destination Format field of the File Manager Control table for the **PRINT** command. A remote name must be the same in both fields. Action: In the File Manager Control table, ensure both names are the same if remote names are specified in both the **Source Format** and **Destination Format** fields.
Unknown configuration option	Cause: An invalid parameter or character is entered in the LAN local names file (**/FLAPP/NET/LOCAL**). Action: Ensure a valid local name is defined on the *LAN* Local Names table. Enter the correct information. See the "FLLAN" on page 227 for information about the LOCAL file syntax.

Chapter 12. . . .
FLLAN

The FactoryLink Local Area Networking (FLLAN) module transmits FactoryLink data between computers (called stations) across a network. A network is a combination of hardware and software that lets multiple computers share resources, such as files, printers, or data. A network consists of the following parts:

- A Network Operating System (NOS)—Software that transports data between software applications on different computers.
- A network application—Software that sends data to a similar application on another computer via the Network Operating System.
- The network hardware—Network interface cards installed on each computer on the network and cables that link them all together.

> **Note:** FLLAN was the first FactoryLink task for sharing data between nodes on a network. In a later version of FactoryLink, the Virtual Real-Time Network and Redundancy (VRN/VRR) task was introduced. VRN/VRR has all of the functionality of FLLAN and is more flexible. FLLAN is still supported, but if you are starting a new application, it is recommended that you use VRN/VRR instead. For more information, see “Virtual Real-Time Network and Redundancy” on page 521.

OPERATING PRINCIPLES

Tags are sent between one station and another using send and receive operations. The tags and operations are defined in the Local Area Network Send and Receive tables. These tables define the conditions under which the send operations are initiated and whether or not the remote station is willing to receive the data.

Sending and Receiving Data

Sending Tag Values to Remote Stations

During a send operation, the FLLAN on the local station sends tag values from the FactoryLink real-time database across the network to the FLLAN on the remote station. The FLLAN on the remote station writes these values to the real-time database on the receiving station.

Receiving Tag Values from Remote Stations

During a receive operation, FLLAN receives values from a remote station and stores them in the FactoryLink real-time database as tags.You do not need the module FLLAN on two or more FactoryLink stations in order to share and store files on a network server or use network printers. External networking software allowing peer services is sufficient to achieve this goal.

Local and Remote Stations

Each station on the network must have a unique station name so FLLAN can identify it. Each station views itself as the local station and views all other stations on the network as remote.

Network Groups

You can combine one or more stations into groups. Grouping permits you to transmit the same data to multiple stations with a single operation. A single station can belong to more than one group. You can use the same group name on more than one remote station; however, these groups are independent and do not correspond to each other.

Using Multiple Platforms on a Network

Because the Network Operating System is transparent to FactoryLink, you can use a different Network Operating System program on each station on a network. This lets you use FactoryLink for different platforms within the same network. You must use the same protocol on all stations in the network.

Monitoring the Network

You can monitor the status of remote stations on the network, such as the number of transmissions the remote station has sent and received and whether these transmissions were successful. You can view the status at run time and other FactoryLink modules can use this information for other activities.

Local Station's Default Values

The local station name and the default values FLLAN uses to transmit data is stored in the local name file **FLAPP/net/local** on each FactoryLink station. We recommend you consult your network administrator if you need to change the default values.

Ordering Tag Names

When sending tag values, FLLAN groups the tags into packets by data type and sends them in the following order: digital, analog, floating-point, message, long analog, and mailbox. To maximize efficiency, place tags of like data-types in the same order in the LAN Send Information table.

Defining TCP/IP Hosts and Service Ports

You must define host and service port information on each station using the following procedure if you are using TCP/IP. If you are using a name server, you may not need to perform some of the steps in this procedure as the name server automatically performs some of these actions.

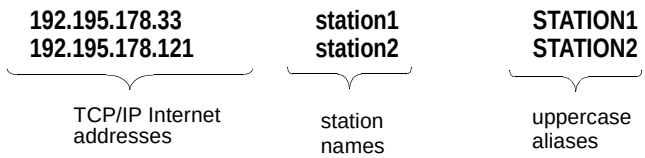
1 Define the TCP/IP Internet addresses for all stations in the hosts file if you are not using a name server or if you are using a name server but the local station name is not in it.

Refer to the vendor's documentation for details on how to modify these files. Contact your system administrator if you do not know your TCP/IP addresses. The syntax for defining the TCP/IP address is

address *sta_name STA_ALIAS*

where

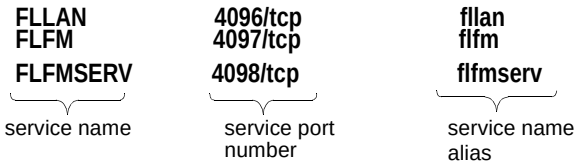
address	Is the TCP/IP internet address.
sta_name	Is the unique name assigned to the station.
STA_ALIAS	Is the alias used to reference the station. This must be in all uppercase. For example,



FLLAN restricts you to 1024 sessions. For each read-only entry in the external domain table, a session is needed for the client and the server. If the entry is a read-write connection, two sessions are created on both the server and client. The 1024 session limit is for each FLLAN application. This means a client can have 1024 sessions and the server can also have 1024 sessions. See the **-n** option for changing the session limit.

2 Define the service ports for FLLAN. The file where you define these depends on your operating system.

Enter the following lines in the file defined for your operating system to define the service ports. Use all uppercase letters for the service names.



Use the service port numbers unless another service name in the services file is already using one of these numbers. If you use different service port numbers, make them consistent for all stations on the network. See the vendor's documentation for details about service port numbers.

FLLAN Wakeup Tag

If data is not sent to remote stations regularly, network connections may time out. You must configure the FLLANSIG tag to prevent network timeouts from occurring.

FLLANSIG is a predefined digital tag that sends a wakeup to the FLLAN task at a configurable interval to all connected remote stations. FLLANSIG also triggers FLLAN to determine whether it is time to attempt connections to unconnected remote stations. FLLANSIG is defined in the Interval Timer Information Table.

FLLANSIG is a number that is less than or equal to the number of seconds in either the TX or CALL parameter, depending on which is less. If you did not change the local station default TX or CALL values, this is a number less than or equal to 10. If you changed the local station default TX or CALL values, this is a number less than or equal to the lesser of the two.

FLLAN does not wake up when the TX or CALL intervals have passed. When the value of FLLANSIG changes, FLLAN wakes up to check whether either of these two intervals have passed.

Testing the Network Connection

After your network is installed, test the communication between stations using the following utilities.

- NR—Tests whether a station can receive data from another station on the network.
- NS—Tests whether a station can send data to another station on the network.

These test programs send and receive data using the same format as FLLAN. Run NR on one station (the local station), and run NS on the remote station to test the communications between two stations on the network. Then, reverse the process for the same two stations. Test every station on the network and test every station as both a local station and a remote station.

Perform the following steps to run the NR and NS test programs:

1 Start NR on the local station. Use the following syntax for this command:

nr *local_name remote_name* [**-d***verbose_level*][**-x***debug_level]*[**-l**] [**-b***bufsize*] [**-a**]

where

local_name Is the name of the computer that receives the data.

remote_name Is the name of the remote computer that sends the data.

verbose_level Controls how much information NR displays about each packet it receives. This can be one of the following:

- 0 Displays the sequence number of messages in multiples of 10 when every 10th message is received. The message is displayed on the same line as the sequence number; the message does not scroll. This is the default.
- 1 Displays the sequence number of the current message. The message is displayed on the same line as the sequence number; the message does not scroll.
- 2 Displays the sequence number of the current message. The message is displayed on different lines and scrolls.
- ≥3 In addition to level 2 output, the message is displayed in hexadecimal format. Any value greater than 3 displays the same information as 3.

debug_level Is a number ≥0 that indicates how much information the network debug layer displays about each packet. The higher the value, the more information NR displays. The default is 0.

-l Writes debug information to a log file named **nr.log** in the current directory.

bufsize Is a number from 128 to 2048 that specifies the number of bytes in a buffer (message). The default is 512.

-a Acknowledges all received messages. If you include **-a** with this command, you must include it with the NS command on the remote station.

In the following example, STATION1 is the local station running NR. STATION2 is the remote station running NS. The local station acknowledges all messages it receives from the remote station.

nr *STATION1 STATION2* **-a**

2 Start NS on the remote station when NR is in the listening mode. Use the following syntax for this command:

ns *local_name remote_name* [**-d***verbose_level*][**-x***debug_level*] [**-l**] [**-b***bufsize*][**-p***secs*] [**-a**]

where

local_name Is the name of the computer that sends the data.

remote_name Is the name of the computer that receives the data.

verbose_level Controls how much information NS displays about each packet it sends. This can be one of the following:

- 0 Displays the sequence number of messages in multiples of 10 when every 10th message is sent. The message is displayed on the same line as the sequence number; the message does not scroll. This is the default.
- 1 Displays the sequence number of the current message. The message is displayed on the same line as the sequence number; the message does not scroll.
- 2 Displays the sequence number of the current message. The message is displayed on different lines and scrolls.
- >3 In addition to level 2 output, the message is displayed in hexadecimal format. Any value greater than 3 displays the same information as 3.

debug_level Is a number ≥0 that indicates how much information the network layer displays about each packet. The default is 0. The higher the value, the more information NS displays.

-l Debug information to a log file named ns.log in the current directory.

bufsize Is a number from 128 to 2,048 that specifies the number of bytes in a buffer (message). The default is 512.

secs Is a number from 1 to 59 that specifies the number of seconds between packet sends.

-a Acknowledges all received messages. If you include **-a** with this command, you must include it with the NR command on the other station.

In the following example, STATION2 is running NS. STATION1 is running NR. STATION2 acknowledges all transmissions from STATION:

ns STATION2 STATION1 -a

After you start NR and NS, they display the following message on the computers they are running on:

n sessions, n buffers, buffer size = n
addname: local_station_name

The programs then display the following message until the two computers establish a connection:

open remote_station_name

You may experience a delay of several seconds between the two messages. Then the computers display the following message:

wait on call

3 Verify the computers establish a connection. After the computers establish a connection, NR and NS automatically begin transmitting messages. The computer running NR displays data-transfer information on its screen each time it receives data. The computer running NS displays data-transfer information on its screen each time it sends data.

4 Press **Esc** on either computer to stop the test.

5 Run NR and NS again at a higher debug level if the computers do not connect. Note any errors that display.

6 Repeat Step 5 until the computers connect.

7 Repeat this procedure again, but run NR on the station you first ran NS and run NS on the station you first ran NR.

8 Repeat this procedure on each station in the network.

NAMING STATIONS AND NETWORK GROUPS

FactoryLink must know the name of each local and remote station on the network running a FactoryLink application as well as any groups defined for the station. This information is defined in Local Area Network Groups configuration table, which consists of two tables:

- LAN Local Names—Defines local station name and any changes to station default values.
- LAN Remote Names—Defines the network groups and stations belonging to a group.

You can fill out as many Send tables as the RAM on your system allows. You can enter as many tags as the available RAM allows. The Local Area Network Send table is filled out in the Shared domain.

The example table defines the following information:

- Names the local station
- Changes the default values of TX and RX
- Defines two network groups and assigns the remote stations to them

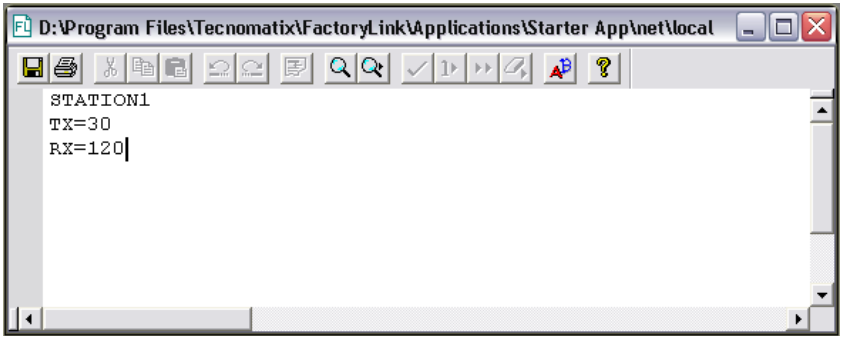
LAN Local Names Table

Local station names are defined in the LAN Local Names table.

Perform the following steps to define the station name for the local computer. You must repeat this procedure for each computer in the network running FactoryLink.

1. In your server application, open **Networking > Local Area Network Groups > local**.
2. Enter the computer name of the local station as defined in the network operating system. (To find out what your computer name is, open the control table and click the Network icon.)

   Computer names are case-sensitive. Enter the computer name in the LAN Local Names exactly as it is spelled in the control table.



3 *(Optional)* To change any of the transmit parameters from their default values, enter the parameters and their new values beneath the station name.

In the example, TX=30 changes the maximum time between data transmissions to 30 seconds, RX=120 changes the maximum time between receipts of data to 120 seconds. The possible transmit parameters and their default values are given below.

4 Press **Enter** at the end of the last line to enter a hard return. If only a station name is entered, then press **Enter** after the station name. **This hard return is required.**

5 Complete the LAN Remote Names table to define the network groups. (See page 239.)

Local Station Transmit Parameters and Default Values

The table below lists all the transmit parameters used by the local station and their default values. The parameters are explained following the table:

Parameter	Default Value
TX	20
RX	60
INIT	0
CALL	10
BUFSIZE	512
MAXLEN	512
MAXSESS	32
ACK	0
ST	10
SD	10

Note: It is recommended that you consult your network administrator before changing any default values.

TX (Transmit Time-out)

A number between 0 and 65,527 that sets the maximum time, in seconds, between transmissions. The default is **20**. If the local station does not send any data to a given remote station after the indicated time, the local station sends an “I am still here” packet to the remote station.

RX (Receive Time-out)

A number between 0 and 65,527 that sets the maximum time, in seconds, between receptions. The default is **60**. **Make sure this value is at least three times greater than the TX value**. If the local station does not receive any data from a remote station after the indicated time, the local station disconnects from the remote station and attempts to reconnect.

If you specify an RX value greater than 60, modify the -t program argument in the System Configuration table; otherwise, FLLANRCV may not shut down properly. To do this, complete the following steps:

1. Open the System Configuration table in the Shared domain. The System Configuration editor appears.
2. Click the right arrow at the bottom of the editor to select the FLLANRCV task.
3. In the Program Arguments field, enter the -t argument with the required RX value. For example, if the RX value in the Local Names table = 90, then enter **-t90**.
4. Click Apply to save the change and then close the System Configuration editor.

INIT

A value of 0 or 1 that specifies whether the local station sends all data when it first connects with another station. The default is **0**. The local station uses this value only when a remote station starts up.

- If INIT = 0, when the local station first connects to another station, it does not send values until one of the values has changed.
- If INIT = 1, when the local station first connects to another station, it sends all values during the first real-time database scan. This can be useful when you start a remote station after the local station has been running. The new station has no values when it starts, so the local station sends the values it has at that time. After that, the local station values are updated normally.

Because startup data can contain uninitialized values, it is recommended that you leave INIT at 0.

CALL

A number between 0 and 65,527 that defines the minimum amount of time, in seconds, the local station waits for a call to a remote station to connect. The default is **10**. If the remote station does not connect to the local station, the local station waits at least CALL seconds before attempting to reconnect. The remote station may still connect to the local station in the interim.

MAXLEN

Only FLLAN uses the MAXLEN parameter. The largest number of bytes a station can send or receive in a single data packet. The minimum is 512; the maximum is 65,536. The default is **512**. The tag data is truncated if a message or mailbox tag is sent that is larger than MAXLEN. Make sure this number is the same on all stations.

- If you enter a value less than the minimum of 512, FLLAN uses 512.
- If you enter a value greater than the maximum of 65,536, FLLAN uses 65,536.

Each tag uses a specific number of bytes, depending on its data type. All tags use 4 bytes to store its tag name + *x* number of bytes to store the value, as shown in the table below:

The tag type...	uses ... for the tag name	+ ... for the value	which =
Digital	4 bytes	2 bytes	6 bytes
Analog	4 bytes	2 bytes	6 bytes
Longana	4 bytes	4 bytes	8 bytes
Float	4 bytes	8 bytes	12 bytes
Message	6 bytes (4 + 2 bytes for the length)	the number of characters in the string	*y* bytes
Mailbox	30 bytes (4 + 26 bytes for the header)	the number of characters in the string	*y* bytes

The MAXLEN parameter must be configured to specify the maximum number of bytes each node requires to send or receive a single data packet.

To distribute alarms and logbook entries along the network, use the following formula to calculate the number of bytes required at each node:

((84 x number of active alarms) + 38) + (number of logbook entries x (24 + msg space)) = bytes

where

number of active alarms	is the maximum number of alarms defined for display in the Active Alarms field in the General Alarm Setup Control table.
number of logbook entries	is the maximum number of logbook entries expected to be generated for the alarms defined. This number can be smaller than or equal to the number of active alarms. A practical estimate of the normal volume of logbook entries is 20-30% of the total alarms.
msg space	this number is smaller than or equal to the number of input lines.

MAXLEN parameters must match on all nodes that receive distributed alarms.

BUFSIZE

Only File Manager uses the BUFSIZE parameter. A number between 128 and 2,048 that sets the size of each buffer in bytes. The default is **512** bytes. The size of the buffer determines the amount of data File Manager can transmit across the network in a single message.

MAXSESS

The maximum number of stations to which the local station can connect at the same time. These are called *connections*. The default is **32**. The maximum number of connections varies by network protocol:

- For NetBIOS, any number from 1 to *x* where *x* is the maximum allowed by NetBIOS. See the NetBIOS documentation.
- For TCP/IP and DECnet, any number from 1 to 64.

ACK

A number from 0 to 1,024 that specifies the number of seconds the local station will wait for a remote station to send a data packet acknowledgment before disconnecting from that station. The default is **0**, which indicates the local does not require an acknowledgment from a remote.

ST (Send Time-out)

A number from 0 to 1,024 that specifies the number of seconds the local station will keep trying to send its data if the remote station cannot accept it because it cannot process data fast enough. The default is **10** seconds. When the time-out expires, the local station generates an error.

SD (Send Delay)

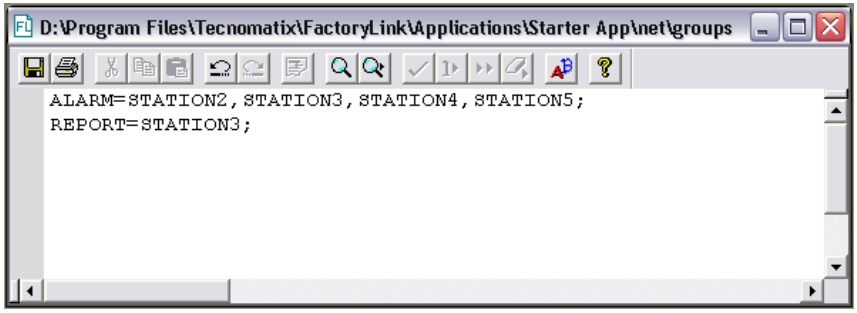
A number from 0 to 1,024 that specifies the number of seconds the local station waits between tries to send its data if the remote station cannot accept it because it cannot process data fast enough. The default is **10** seconds. If you increase this number, you will reduce CPU consumption but you may cause the overall performance to drop.

LAN Remote Names Table

Network groups are defined in the LAN Remote Names table. You must assign each remote station on the network to a network group. Do not include the name of the local station in a network group.

Perform the following steps to define network groups for the local station. You must repeat this procedure for each computer in the network running FactoryLink.

1 In your server application, open **Networking > Local Area Network Groups > Groups**.



2 Complete the LAN Remote Names table. Enter all group names on a separate line using the following format.

group **=** *STATION***[,** *STATION*,...**];**

where

group	Alphanumeric name of 1 to 16 characters assigned to a network group. Group names can be either upper or lowercase.
STATION	Computer name, as defined in the network operating system, of one or more remote stations to include in the group. Spell the name of the remote station as it is spelled in the control table. Separate each name with a comma but no space. The station list can be split at the end of the line, but the comma must be on the same line as the station name. Place a semicolon (;) at the end of a group name definition.

In this example, the ALARM group consists of STATION2, STATION3, STATION4, and STATION5. The REPORT group consists of STATION3. Note that STATION3 belongs to both groups and that each line ends in a semicolon.

3 Press **Enter** at the end of the last line to enter a hard return. **This hard return is required.**

4 Click **Save** and close this table.

SENDING TAG VALUES TO OTHER STATIONS

Data is sent from one station to another using send operations. Each send operation defines which data to send when the operation executes. The operations and the data to send with each operation are defined in the Local Area Network Send configuration table.

This chapter provides a field-by-field description of the following two tables:

- LAN Send Control table—Defines the stations to receive data and the conditions under which FLLAN sends the data.
- LAN Send Information table—Defines the tags sent to the remote stations with the send operation.

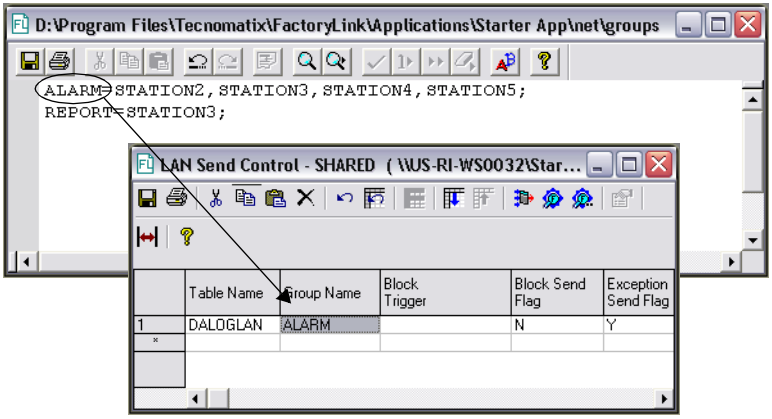
You can complete as many Send tables as the RAM on your system allows. You can enter as many tags as the available RAM allows.

LAN SEND CONTROL TABLE

Send operations are defined in the LAN Send Control table.

Accessing

In your server application, open **Networking > Local Area Network Send > LAN Send Control.**



Field Descriptions

Specify the following information for this table. Add an entry for each send operation you want FLLAN to transmit across the network.

Table Name Name to reference the send operation.

Valid Entry: up to 16 alphanumeric characters

Group Name Name of the group of network stations that the local station sends data to. This name must match a group name defined in the LAN Remote Names table.

From the example used for the LAN Remote Names, the group ALARM is defined as STATION2, STATION3, STATION4, and STATION5. Therefore, the FLLAN task on the local station (STATION1 from the LAN Local Names example) would send data to the stations belonging to the ALARM group.

Enter ALL if you want to send the data to all stations named in the GROUPS file.

Valid Entry: up to 16 alphanumeric characters

Block Trigger Name of a tag that triggers this operation. When the change-status flag for this task is set, FLLAN sends the values specified in the LAN Send Information to the stations included in the group specified in the **Group Name** field of this table. You must activate the send operation with the **Exception Send Flag** field if you leave this field blank.

You can specify both a trigger in this field and an exception send in the **Exception Send Flag** field on this table. The operation executes each time the trigger is set and each time the tag value changes; however, if you do this, the value may not always be sent on exception because the block trigger may reset the change status bit before FLLAN processes the exception send table.

The tag specified must be in the Shared domain.

To send the table only when this tag value is forced to 1 (on), define this tag as digital.

To send this table whenever this tag value changes, define this tag as analog, longana, floating-point, message, or mailbox.

To send individual values only as the table tag values change, rather than as a triggered block send, leave this field blank.

You must define the **Block Send Flag** if you define this field.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message, mailbox

Block Send Flag

Values sent when this operation is triggered. This can be one of the following:

W Writes only the values that have changed since the last send.

YES Force-writes only the values that have changed since the last send. If you enter YES, FLLAN does not send a tag's default value at start up. Instead, FLLAN waits to send it until the tag value changes to something other than the default.

NO Sends all values except empty mailboxes, whether or not they have changed.

You must define a **Block Trigger** for the **Block Send Flag** to work.

Exception Send Flag

Defines whether or not to send values of individual tags as they change. This can be one of the following:

W Writes values that change when they change.

YES Force-writes values that change when they change. If you enter YES, FLLAN does not send a tag default value at start up. Instead, FLLAN waits to send it until the tag value changes to something other than the default.

NO Does not send values when they change. If you enter NO, you must activate the send operation with the **Block Trigger** field.This is the default.

You minimize network traffic if this data is changing infrequently to send only the values that change. If data is changing frequently and at regular intervals, then it is more effective to send the data in triggered blocks.

You can also specify both a trigger in the **Block Trigger** field and an exception send in this field for the same operation. The operation executes each time the trigger is set and sends the individual tags each time a tag value changes.

If you set up FLLAN to send the data as it changes and the change bit for the data is set before FLLAN establishes communication with the remote node, FLLAN does not send the data until the next time the data changes. Define a Send State tag in the Network Monitor table (see "Monitoring

Remote Station Communications" on page 247 for more information) and define a Math & Logic procedure to monitor the value of the **Send State** tag to ensure FLLAN sends the changed data.

Do not use the same tag in more than one exception table if you specify exception sends. If you have the same tag in a block send table and an exception send table, the triggered table may reset the change status bit before the exception send table is processed with the result that the value/tag is not sent on exception.

Note: For further explanation, the following table lists the flags to use based on what modes to use.

Block Trigger	Block Send Flag	Exception Send Flag	Flag
sec1	NO	NO	1
sec1	W or YES	NO	2
none required	W or YES	W or YES	3

Enable/Disable Tag

Name of a digital tag to disable this operation. When the value of this tag is set to 0, this operation is not executed, even when the **Block Trigger** is set. This field disables the operation for all stations included in the group.

This tag is useful if a remote station will not be available for the network for a long period of time.

Valid Entry: tag name
Valid Data Type: digital
Default: 1

LAN Send Information Table

The data to send with each send operation is defined in the LAN Send Control table.

Accessing

Position the cursor on the line entry on the LAN Send Control table representing the send operation you are configuring. In your server application, open **Networking > Local Area Network Send > LAN Send Control > "your table name" > LAN Send Information**.

Field Descriptions

Tag Name

Tag to send with this operation. The tag specified must be in the Shared domain. Group tags by data type and order them by digital, analog, floating-point, long analog, message, and mailbox if you want to maximize performance.

If you enter message or mailbox tags, ensure the MAXLEN value is set to slightly larger than the longest message or mailbox value. The value is truncated if the MAXLEN value is set lower than the length of a message or mailbox value.

Valid Entry: tag name

Valid Data Type: digital, analog, longana, floating-point, message, mailbox

Network Alias (Optional)

Alias name that FLLAN uses to transfer data on the network. The alias name is a tag name used globally by all stations on the network. It identifies data being sent from one station to another. Define this name on the sending station and reference it on all remote stations that receive data from the sending station.

If you leave this field blank, FLLAN automatically uses the name in the **Tag Name** field as the network alias.

If you use an alias name, you can be more flexible when naming tags among systems. For instance, at one station an analog tag may be called **alrm7**, while at another station an analog tag containing the same data may be called **temphigh**. To transfer data across the network from **alrm7** to **temphigh**, you can designate an alias name, such as **hot**, for that data.

Valid Entry: up to 48 alphanumeric characters (Do not use a number for the first character.)

For example, the local station sends the value of the tag **regular_tank_level** to a corresponding tag on some remote station. The tag on the remote station may, or may not, be the same name**.** If it is not the same name, then you can specify an alias to link the “sending” tag on the local station to the “receiving” tag on the remote station.

Using **r87_tank_level** as the name of the receiving tag on the remote station, the local station sends the value of **regular_tank_level** across the network identified as **tank_level.** The alias **tank_level** would also be used on the remote station and map to the tag **r87_tank_level**. Because the alias between the two stations is the same, the local station can send the value of **regular_tank_level** to the remote station tag **r87_tank_level**.

Receiving Tag Values from Remote Stations

FLLAN can receive data from tags on remote FactoryLink stations and write them to the Real-Time Database on the local station. This mapping is defined in the Local Area Network Receive configuration table.

This chapter provides a field-by-field description of the following two tables:

- LAN Receive Control table—Defines the remote stations that can send data to the local station.
- LAN Receive Information table—Defines the tags in the local real-time database that are updated by data received from remote stations. Another station cannot change its value unless you list a tag name in this table. This provides security for sensitive data.

You can complete as many Receive tables as the RAM on your system allows. You can enter as many tags as the available RAM allows.

LAN Receive Control Table

Remote stations that send data to this local station are identified in the LAN Receive Control table.

Accessing

In your server application, open **Networking > Local Area Network Receive > LAN Receive Control.**

Field Descriptions

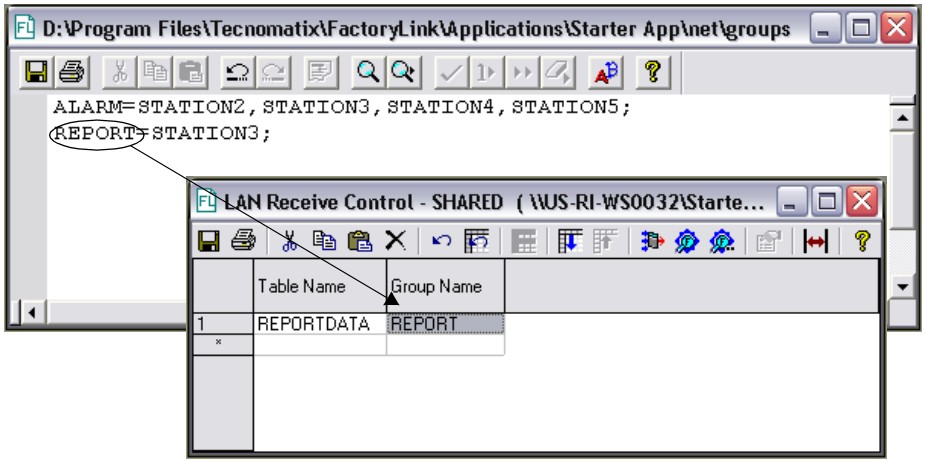
Table Name — Name for this receive operation.

Valid Entry: up to 16 alphanumeric characters

Group Name — Name that identifies the group of network stations where FLLAN receives the tag values. This name must match a name defined in the LAN Remote Names table.

Enter **ALL** if you want to receive data from all stations named in the GROUPS file.

Valid Entry: up to 16 alphanumeric characters



In this example, the local station receives data from the remote stations belonging to the network group REPORT.

LAN Receive Information Table

Define the tags in the Real-Time Database for the local station that are updated with the data received from a remote station.

Ensure the cursor is positioned on the line entry in the LAN Receive Control table representing the receive operation you are configuring.

Accessing

In your server application, open **Networking > Local Area Network Receive > LAN Receive Control > "your table name" > LAN Receive Information.**

Field Descriptions

The name of the receive operation you are configuring is displayed in the **Table Name** field at the bottom of the table.

Specify the following information for this table. Add an entry for each tag received from any remote station in the network group.

Tag Name — Tag to be updated when the data identified by network alias is received. The tag specified must be in the Shared domain.

If you enter message or mailbox tags, ensure the MAXLEN value is set to slightly larger than the longest message or mailbox value. The value is truncated if the MAXLEN value is set lower than the length of a message or mailbox value.

Valid Entry: tag name

Valid Data Type: digital, analog, longana, float, message, mailbox

Network Alias (Optional)

Alias name that FLLAN uses to transfer data on the network. This name must match the name assigned to a database tag in a send operation on a remote station.

The alias name is a tag name used globally by all stations on the network. It identifies data being sent from one station to another. Define this name on the sending station and reference it on all stations that receive data from the sending station.

If you leave this field blank, FLLAN automatically uses the name in the **Tag Name** field as the network alias.

If you use an alias name, you can be more flexible when naming tags among systems. For instance, at one station, an analog tag may be called **alrm7** while at another station an analog tag containing the same data may be called **temphigh**. To transfer data across the network from **alrm7** to **temphigh**, you can designate an alias name, such as **hot**, for that data.

Valid Entry: up to 48 alphanumeric characters (Do not use a number for the first character.)

MONITORING REMOTE STATION COMMUNICATIONS

Configure the Network Monitor table if you want to monitor the status of remote stations on the network, such as the number of transmissions a remote station has sent and received and whether these transmissions were successful. This table consists of the Network Monitoring table that defines the stations you want to monitor and the tags to hold the information.

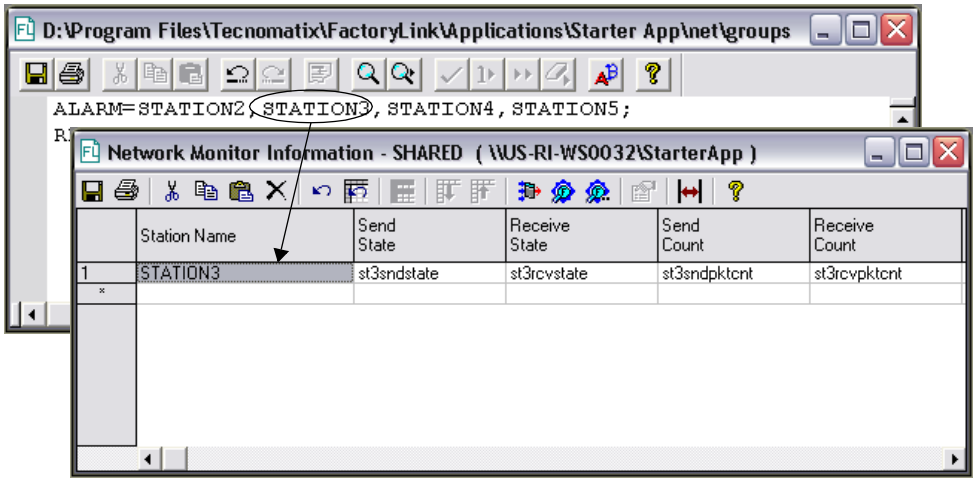
You can monitor any or all FactoryLink stations on the network as long as they are running FLLAN. The example table defines tags to contain the status of the STATION3 remote station.

Accessing

In your server application, open **Networking > Network Monitoring > Network Monitor Information.**

Field Descriptions

Station Name — Name of the remote station to monitor. This name must match one of the station names in the LAN Remote Names table.



Send State — Tag to contain the status of transmissions to the remote station. At run time, FLLAN updates this tag with values to indicate the status:

- 0 The remote station is available.
- 1 The remote station is not available and is not yet active.
- 3 The local station called the remote station but the remote station has not responded.
- 6 One of the stations has disconnected.
- 7 The connection has terminated, because the remote station buffer is a different size from the local station buffer.
- 10 The remote station is ready to connect.
- 11 The remote station responded to a call request but it is not yet ready to receive data.
- 12 The remote station is ready to receive data. Use this value to ensure you have established communication with the remote node before sending any data.

Valid Entry: tag name
Valid Data Type: analog

Receive State Tag to contain the status of transmissions from the remote station. At run time this tag uses the following values to indicate the status.

0	The remote station is available.
1	The remote station is not available and is not yet active.
2	The local station is listening for a call from the remote station.
6	One of the stations has disconnected.
7	The connection has terminated. This occurs if FLLAN does not reset the remote station when it disconnects.
10	The remote station is ready to listen.
11	The remote station called the local station but the local station is not yet ready to receive data.
12	The local station is ready to receive data.

Valid Entry: tag name
Valid Data Type: analog

Send Count Tag that counts the number of times the local station sent data to the remote station. FLLAN does not reset this value to 0 when the remote station disconnects.

Valid Entry: tag name
Valid Data Type: analog

Receive Count Tag that counts the number of times the local station received data from the remote station. FLLAN does not reset this value to 0 when the remote station disconnects.

Valid Entry: tag name
Valid Data Type: analog

Send Sequence Tag that indicates the number of the packet the local station sent.

FLLAN assigns a number, called the send sequence number, for each packet the local station sends to the remote station you are monitoring. As the local station sends each packet, this number increments. You can monitor this number to see whether the local station is sending data packets in the correct order. FLLAN resets this value to 0 each time the stations reconnect.

Valid Entry: tag name
Valid Data Type: analog

Receive Sequence	Tag that indicates the number of the packet the local station receives. FLLAN assigns a number called the send sequence number for each packet the local station sends to the remote station you are monitoring. As the local station receives each packet, this number increments. You can monitor this number to see whether the local station receives data packets in the correct order. FLLAN resets this value to 0 each time the stations reconnect.
Sequence Errors	Tag that counts the number of times the local station sends or receives data out of sequence. Valid Entry: tag name Valid Data Type: analog
Send Errors	Tag that counts the number of times the local station could not successfully send data to the remote station. This usually occurs when the local station tries to send data to the remote station while the remote station is disconnected. Valid Entry: tag name Valid Data Type: analog
Receive Errors	Tag that counts the number of times the local station could not successfully receive data from the remote station. This usually occurs when the remote station is disconnected and the local station is waiting for data. Valid Entry: tag name Valid Data Type: analog
Send Error Message	Name of a message tag that contains the latest error or status message for the send link for the remote station. Valid Entry: tag name Valid Data Type: message
Receive Error Message	Name of a message tag that contains the latest error or status message for the receive link for the remote station. Valid Entry: tag name Valid Data Type: message

To view the status on screen at run time, design and configure a graphics screen to display the information.

If you want other FactoryLink modules to view and use this information for other activities, configure those modules' tables. For example, you can configure Math & Logic and Alarm Supervisor to monitor these tags and trigger an alarm whenever a remote station disconnects.

PROGRAM ARGUMENTS

Argument	Description
-D<#>	Set verbose level. (# = 0 to 22)
-L	Enables logging of debug information to a log file.
-R (LAN Send only)	Prevents setting LAN Send Enable/Disable tag to 1.
-T	Insert timestamp at beginning of each debug statement.
-S<#>	Closes and reopens log file every # messages.
-W<#>	Wraps log file every # messages.
-X	Logs underlying network software's error messages to a log file.

Chapter 13 . . . .
Historians

The Historian task is the interface between FactoryLink and a relational database. It processes data requests from other FactoryLink tasks and sends them to the relational database. Data requests from Database Logger or Data Point Logger tasks can store data in the relational database. Data requests from Trending or Database Browser tasks can retrieve data from the relational database.

There are historians for various relational databases, including:

- ODBC (used for SQL Server and other relational databases with ODBC support)
- Oracle
- Sybase
- dBASE IV

OPERATING PRINCIPLES

The following steps describe how a historian processes data requests for a relational database:

1. A FactoryLink task sends a data request to a mailbox historian service. This can be a request from Database or Data Point Logging to store data in the relational database or from a task like Trending to retrieve data from the relational database. FactoryLink tasks submit their requests for data in the form of Structured Query Language (SQL) statements. Generally, mailboxes are unidirectional: a task requesting data from the historian makes the request through a different mailbox than the mailbox historian uses to return data.

2. Historian reads this mailbox and processes any queued data requests. It transmits the data request to the relational database server.

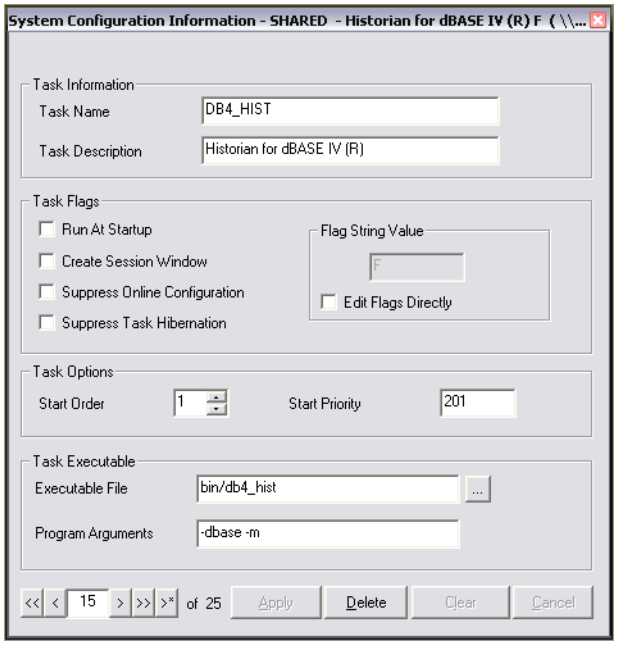
3. The relational database returns the requested information to the historian if the request was to retrieve data.

4. Historian returns the requested data to the requesting task.

SETTING HISTORIAN RUN-TIME PARAMETERS

Perform the following steps to configure a historian as part of the FactoryLink application. You perform this procedure for each historian you want to configure.

1 In your server application, open **System > System Configuration > System Configuration Information** in the form view.



2 Choose the type of historian you want to configure:

- For dBase IV and ODBC, browse through the System Configuration Tasks using > until the desired task appears in the Task Description box.
- For Oracle and Sybase, create a new task using >* and perform these steps:

  1) In the Task Name box, type:

     For Oracle, **OR8_HIST, OR9_HIST,** or **OR10HIST**(depending on your version).

     For Sybase, **SYB_HIST**.

  2) In the Task Description box, type a description for the respective database: **Historian for Oracle** or **Historian for Sybase**.

3) In the Executable File box, type:

   For Oracle, **bin/or8_hist, bin/or9_hist,** or **bin/or10hist**(depending on your version).

   For Sybase, **bin/syb_hist**.

3 In the Program Arguments box, type the desired arguments. See a list of the program arguments on page 280.

4 Under Task Flags, select the **Run At Startup** check box. Click **Apply** and exit.

ODBC HISTORIAN OVERVIEW

The ODBC ((Microsoft) Open DataBase Connectivity) historian allows FactoryLink to access data from multiple, diverse Relational DataBase Management Systems (RDBMS). ODBC focuses on the Application Programming Interface (API) that supports connections to various database systems. The standard used for connecting and accessing a database is the Structured Query Language (SQL). Software components, called drivers, link a FactoryLink application to an RDBMS. The ODBC historian allows multiple instances of the task to run in a single application.

The ODBC historian enables FactoryLink to access data from several diverse database systems through this single interface while the historian remains independent of any RDBMS from which it accesses data.

The ODBC interface defines the following:

- Dynamic-link libraries (DLL) of ODBC function calls to connect to an RDBMS, execute statements, and retrieve results
- SQL syntax of function calls
- Standard representation for data types and the mapping between FactoryLink and RDBMS data types
- Error codes

The following components work together to make ODBC and FactoryLink communicate:

- FactoryLink—Performs processing and calls to third-party ODBC drivers to provide data to or request data from a data source.
- Driver Manager—Loads ODBC drivers for the needed data source.
- Driver—Processes ODBC function calls, submits SQL requests to a specific data source, and returns results to FactoryLink.
- Data Source—Contains the data the driver accesses. Connection strings link a data source to a driver.

Considerations

Supported Drivers

The ODBC historian supports drivers for Windows platforms. These drivers handle the connections to the various platforms relational databases run on.

ODBC defines two types of drivers:

- Single-tier—Processes both ODBC calls and SQL statements
- Multiple-tier—Processes ODBC calls and passes SQL statements to the data source

Single-tier drivers do not require additional RDBMS software; however, multiple-tier drivers do require the purchase of RDBMS server and connectivity products. Communication with most RDBMSs on servers requires the installation of the RDBMS client software on the FactoryLink client.

The following table specifies the required additional connectivity software for each driver. For specific information regarding the additional software requirements, refer to the document on the specific driver and the network protocol that connect to your server. For specific software version numbers for the various products listed in the table below, see the *Installation Guide*.

Driver	Type	Bit	Additional Software Requirements
Access	Single-tier	32-bit	None
MS SQL Server	Multiple-tier	32-bit	MS SQL Server Client
Sybase	Multiple-tier	32-bit	Sybase Open Client-Library and Net-Library

Conformance Levels

Drivers and their associated RDBMS provide a varying range of functionality. The ODBC historian requires that drivers conform to the Level 1 API conformance, which determines the ODBC procedures and SQL statements the driver supports. Use of the Level 2 API function **SQLExtendedFetch** is based on whether the driver and its data source support it.

SQL Statements

The ODBC historian does not totally depend on the SQL conformance levels, but rather it tries to map the FactoryLink data types to the best match provided by each data source. When a data type maps, its SQL statement is accepted as long as the driver and data source can perform that operation.

Data stored on an RDBMS has an SQL data type, which may be specific to that data source. A driver maps data source-specific SQL data types to ODBC SQL data types and driver-specific SQL data types.

Data Types Supported by ODBC Driver

For information on the data types supported by the relational database, with which the ODBC driver is transacting, refer to the database documentation.

In the previous versions of FactoryLink, the format of the FactoryLink supported date data type had to be a string in the format of *yyyymmddhhmmss*. The tag had to be defined as a message with a minimum default length of 14 bytes to retrieve the date data type into a tag. If a tag was to insert or update a database row with the date data type, then the tag also had to be a message data type using previous format. In addition, the product now supports a direct conversion to and from a long analog tag (such as SECTIME) that equates to the elapsed seconds since January 1, 1980 to the date data type.

Conversion Issue

If you are converting an application that has the ODBC historian configured, the conversion to the multi-instance ODBC historian requires that you run FLCONV directly against the pre-6.6 ODBC historian configuration or a restore of a platform-specific save. Do not perform a multiplatform restore of the application before running FLCONV.

Setting Up ODBC

The general steps for setting up ODBC with FactoryLink are:

1. Install a driver for each RDBMS FactoryLink should access.
2. Connect a data source to the driver.
3. Set up driver sources.
4. Complete the necessary FactoryLink Historian Mailbox tables.

The detailed instructions for setting up these steps are described in detail in the following sections.

Setting up ODBC Drivers and Data Sources

The in-depth information regarding installing a supported driver, its data sources, and their network connections are not within the scope of this guide. Refer to the documentation associated with the specific driver you want to install.

Drivers and Data Sources

Use the ODBC Administrator to add and delete drivers, and to add, configure, and delete data sources.

Perform the following steps to complete information for drivers and data sources:

1. Open the ODBC Administrator from the Control Panel. Click the ODBC icon.

   After you install an ODBC driver, define one or more data sources for it. A data source name provides a unique pointer to the name and location of the RDB associated with the driver.

   The data sources defined for the currently installed drivers appear in the User DSN box in the ODBC Data Source Administrator dialog box.

2. Select the driver you want to define as part of the data source definition.

3. Click Add to display the setup dialog for the selected driver.

4. Type the **DSN** (Data Source Name). This same name must be used in the Historian for ODBC Information table explained in the “Historian Information Table” on page 262.

5. For the setup instructions on each supported driver, see “Defining Drivers” on page 258.

Defining Drivers

Within this section are subsections for each driver you can define and the syntax for the data source you must enter in that screen and on the ODBC Historian Information table.

SQL Server Driver and Data Source

The SQL Server driver supports the SQL Server database system available from Microsoft and Sybase.

Perform the following steps to complete setting up the SQL Server:

1. Enter the server name (where Microsoft SQL server database is located).

2. Enter the **Database Name** and then select **Two Phase Commit** when prompted.

3. Click **OK** to add the Data Source Name.

4. Click **OK** to close the ODBC Administrator.

Microsoft Access Driver and Data Source

For detailed information, refer to the *Microsoft ODBC Desktop Database Drivers Getting Started* guide. Perform this procedure to set up the Microsoft Access Driver and Data Source:

1. To create a new database, click **Create**. Choose the path drive and directory, such as **d:\fl660acc97**, and database name, such as **plant1.mdb**. Click **OK** and the database is created, which a popup message indicates.
2. To connect to an existing database, click **Select** and then **OK** for the path and database file.
3. Click **OK** to accept this setup. Then, click **OK** to close the ODBC Administrator.

Sybase System Driver and Data Source

The Sybase System driver supports the SQL Server 10 database system available from Sybase. For information on the setup information you must enter, refer to the *MERANT DataDirect ODBC Drivers Reference* guide. Perform this procedure to complete setting up the Sybase System 10 driver and Data Source:

1. Type the Server Name (from the SyberClient software that is already installed on the FactoryLink Client computer).
2. Type the Database Name, then click **OK** to add the Data Source Name.
3. Click **OK** to close the ODBC Administrator.

Historian Configuration Tables

A single historian task can service several client tasks such as the Database Logger, Database Browser, Trending, Alarm and Data Point Logger tasks. However, a single historian can only process one database query at a time. This design can result in application performance issues if not handled correctly, since some queries can be time-consuming and block the execution of more critical queries.

The ODBC historian supports the configuration and execution of up to 10 instances of the task in a single application. This allows developers to selectively distribute the various database queries required by the application across different running instances of the task. The developer can route the more critical and high-speed queries to one historian instance and the slower and less critical requests to another and thereby alleviate the performance issues associated with a single historian servicing all client queries.

For example, the execution of stored procedures through PowerSQL, large SELECT, UPDATE, or DELETE queries from DBBROWSE and PowerSQL and Historical data requests by Trending have the potential to be time-consuming queries. However, the logging of records

by the Database Logger or Data Point Logging tasks are generally a faster and more time critical operation. Therefore, one instance of the ODBC historian could be configured and run to service all Database Logger queries and another to handle the PowerSQL and DBBROWSE task queries.

The configuration for a trend chart using the Real-Time and Historical Trend Control requires that the logging be routed to the same historian used by the trending task. The Multi-instance ODBC historian still has potential for performance relief in this situation but would require a slightly more complex configuration. One possibility is to use the Real-Time Trend Control if you do not need historical data. Another possibility is to configure one chart for real-time only that uses logging and trending through one historian instance, and another chart just for historical viewing through another historian instance. The distribution can also be set up so that some of the queries from a specific client go to one historian instance while others are routed to another instance.

Database queries are routed to a specific historian by defining a unique mailbox (or set of mailbox) tags and database data source names for each instance and referencing these mailbox tags and data source names in the ODBC task configuration tables.

The following rules apply to the configuration requirements across the historian instances:

1. Each instance of the ODBC historian *must* use a unique set of mailbox tags.
2. Each instance of the ODBC historian *must* use a unique set of Disable/Enable Connection, Connection Status, and Database Error tags. If tags are used for the Connection String, they must be unique for each historian instance.
3. Alias names could be the same in different instances of the historian since these are simply symbolic references. However, this could cause confusion in debugging and interpretation of run-time messages by operators; so it is discouraged.
4. Different instances of the historian may reference the same connection strings and thus connect to the same database. However, connections to the same databases from different historian instances will result in additional physical connections and must be considered when configuring the user license requirements for database servers. Multiple references to the same connection strings within one historian instance do not create multiple physical connections to the database.

Historian Instance Number Table

The Historian Instance Number table is used to specify the instance numbers to be configured. The instance numbers entered create a relationship to the specific configuration information required by that instance in the Historian Mailbox Information and Historian Information tables. (Running the Convert utility on an application that contains an ODBC historian configuration updates the configuration tables and defines the existing configuration to instance zero.)

Accessing

In your server application, open **Historians > Historian for ODBC > Historian Instance Information for ODBC.**

Field Description

Historian Instance ID — Enter a number to specify the specific instance of the ODBC historian task being configured. The first instance to be configured is instance zero.

Create a separate mailbox or set of mailboxes for each instance of the historian that is to be configured. Different historian instances may not reference the same mailboxes.

Valid Entry: 0 to 9

Historian Mailbox Information Table

Historians process data requests queued to a mailbox. A single historian can service multiple mailboxes, however, there is no requirement to create multiple mailboxes for an instance of the historian. There is also no advantage in using multiple mailboxes from an application behavior or performance standpoint. It is generally more efficient to use a single mailbox for each Historian instance, regardless of the number of clients it services, than to have multiple mailboxes. However, some developers may choose to use different mailboxes for purposes related to application maintenance and readability. The mailboxes a Historian services are defined in the Historian Mailbox Information table.

Accessing

In your server application, open **Historians > Historian for ODBC > Historian Instance Information for ODBC > ”your instance ID name” > Historian Mailbox for ODBC.**

Field Description

Historian Mailbox — Mailbox this Historian services. This name must match the name defined in the task using Historian to process data requests.

Create a separate mailbox or set of mailboxes for each instance of the Historian that is to be configured. Different Historian instances may not reference the same mailboxes.

Valid Entry: tag name
Valid Data Type: mailbox

Historian Information Table

Database connection information is defined in the Historian Information table. Add at least one entry for each uniquely defined database to be accessed by this Historian instance. Database connection information may be shared by multiple client tasks. Multiple entries for the same database connection are only necessary if separate control and monitoring of the connection status (and the tables that define) are required through the use of the Disable/Enable Connection, Connection Status, and Database Error tags. If the connection to a database via its defined alias closes for one task, it will close for all tasks. If separate entries are defined, then closing the connection for one alias does not affect the tasks using the other entries.

Accessing

In your server application, open **Historians > Historian for ODBC > Historian Instance Information > "your instance ID name" > Historian Information for ODBC**.

Field Descriptions

Database Alias Name
: Unique name to represent a database connection. This must match the database name defined in the client task using Historian to process data requests.

  Valid Entry: database connection name

Disable/Enable Connection
: Tag that enables or disables the connection. When this tag is set to 1, the connection to the relational database defined in this entry is closed; when set to 0, the connection opens.

  Valid Entry: tag name
  Valid Data Type: digital

Note: Database aliases should not share connection tags. Sharing connection tags between database aliases can result in errors.

*Connection String
: String required to connect to the database. The connection string information defined in an ODBC driver setup must match what you define in the ODBC Historian Information table. Use either a short DSN or a long DSN in the connection string as defined below.

  DSN=*data_source_name*

  where

  data_source_name is the Data Source Name defined in the ODBC Data Source Administrator

  for example, DSN=Access7 or

  DSN=*data_source_name*[;*attribute*=*value*[;*attribute*=*value*]..]

where

[;*attribute= value...*]

provides optional pairs of information, such as a user ID and password, used when more information or overrides are needed to log on. The default values are those set in driver dialogs.

for example, DSN=Oracle_NT; UID=flink; PDW=flink

Information specified in a connection string either adds to or overrides the data source information defined in the **Setup** dialog for each driver. This can either be a constant or a tag name. If you enter a constant, precede the connection string with a single quote.

You must specify the connection string in the tag **Default Value** field and a length in the **Length** field that accommodates the longest connection string you might define for this tag if you enter a tag name.

If the connection string exceeds the defined length, it is truncated and the connection will not be made. Define 254 as the length for the best results.

Valid Entry: message or string constant of up to 254 characters

To connect to a local Oracle database, enter the connection string as:

DSN=data_src;SRVR=;UID=userid;PWD=password

Connection Status — Tag that is updated by the Historian that defines the state of this connection.

Note: Database aliases should not share status tags. Sharing status tags between database aliases can result in errors.

Valid Entry: tag name
Valid Data Type: analog

Database Error — Tag to receive the error value passed from the database software.

The tag should correspond to the type of error the relational database sends. If it is a number use long analog; if it is text use message.

Valid Entry: tag name
Valid Data Type: longana or message

Note: The **Database Error** tag is updated only when a fatal error is defined in the **flhst.ini** file or if a database open connect call fails.

Executing Multiple Instances of ODBC Historian

The following procedures are required to execute multiple instances of the ODBC Historian:

1. An entry must be added to the System Configuration table for each instance of the Historian to be executed.
2. The first instance to run is always considered instance zero. Additional instances are 1 through 9, for a total maximum of 10 instances.
3. The Task Name filed for instance zero (the first instance) must be ODBCHIST. This is the same as in previous versions; existing applications do not require any modification to the System Configuration table. The FLCONV function will make all necessary modifications. The Task Name for each additional instance to be added is ODBCHISTn, where n is the instance number (1-9).
4. The Program Arguments field in the System Configuration table should include a new argument -Un, where n is the instance number (0-9). The argument is not required for the first instance; if omitted, -U0 is assumed. The argument is required for all other instances.
5. The entry in the Executable File field of the System Configuration table is bin/odbchist for all instances.

> **Note:** The correct operation of FLCONV utility to convert the old ODBC Historian tables to new multiple instance ODBCHIST tables should be run on the earlier version of the application, which has ***not*** been restored with the current FLREST option. To transfer an earlier version application from another computer, you need to have it in the form of that version's platform specific save file, or a .zip file or similar format.

Administrative Duties for ODBC Historian

Some of the ongoing administrative duties required for ODBC are:

- Adding and deleting drivers through the ODBC Administrator. You can open the ODBC Administrator either from the Control Panel or from the ODBC program group.
- Adding, modifying, and deleting data sources through the ODBC Administrator program.
- Testing unsupported drivers using the ODBC driver conformance utility.

Maintaining Drivers and Data Sources

The Microsoft ODBC Desktop Database Drivers diskettes and documentation are included with your ODBC Historian. Refer to the ODBC *Getting Started* manual regarding Access Drivers or the *MERANT DataDirect ODBC Drivers Reference* book regarding Oracle, the SQL Servers, and Sybase System drivers to set up your ODBC drivers.

Testing Unsupported Drivers

The ODBC Driver Conformance Test utility validates the level of conformance provided by an unsupported driver meets the requirements of a supported data source. This utility is installed by default in the **FLINK/BIN** directory during installation. This directory contains all the FactoryLink program files. The executable program file for this utility is **FLHSTDRV.EXE**.

Before you start, a driver must be connected to a data source.

Perform the following steps to use the ODBC Driver Conformance Test utility:

1. Run the utility executable: **FLHSTDRV.EXE** to display the **Data Sources** dialog listing the SQL data sources already set up through the ODBC Administrator.
2. Choose a data source from the displayed list, then click OK. A message notifies you of a successful connection to the data source.
3. The FactoryLink Driver Conformance Test window is displayed behind the message. From this window, the File pull-down menu lists the options:
   - Connect
   - Disconnect
   - FactoryLink Driver Conformance Test
4. Choose the Driver Conformance Test option to run the test and display the test results:
   - Successful—FactoryLink Auto-Test message confirmation of
     Driver PASSED minimum FactoryLink **conformance requirements**
   - Unsuccessful—Driver is not supported.
5. Click **OK** to display a chart that gives status details from the test.

If the test is unsuccessful, disconnect the current data source from the **File** menu and connect it again. Or, you may want to connect to a different data source for another test.

> **Caution:** Passing the FactoryLink Driver Conformance Test means that an ODBC driver passed only the minimal FactoryLink conformance requirements and it may work with the ODBC Historian. However, a driver could pass the test and still be incompatible. There is no brief test available to certify that a driver is supported completely. Testing of ODBC drivers is performed with each release and a list of the drivers that were tested and certified for use with the release is provided in the *Installation Guide*.

ORACLE HISTORIAN

This section provides information needed to configure the Oracle Historian.

Considerations

This section explains how to set your FactoryLink application to work with the Oracle historian. Read this section before you configure your historian.

Network Software Needed

If you want to use the Oracle historian, refer to the release notes for the Oracle-specific software to use.

Oracle Licenses

Oracle requires you to purchase licenses for the number of FactoryLink processes using an Oracle database. Connection strings often use platform-defined aliases to reference Oracle servers.

Calculate the number of Oracle licenses required for each Oracle database:

1. One license for the Database Administrator
2. One license for each unique Oracle User Name and connection string pair

The minimum user requirement to connect FactoryLink to one Oracle server is two user licenses.

> **Note:** Each historian running in the application creates a FactoryLink process. For example, there are four historians (ODBCHIST, ODBCHIST1, ODBCHIST2, and ODBCHIST3) on a FactoryLink client computer 1, all talking with the Oracle server computer, while on client computer 2, there are also four historians talking with the same server computer. Even if all eight historians use the same user name and password, for example, flink/flink, the server considers these as eight different processes. For license information, check with Oracle.

Number of Cursors per User

The OPEN CURSORS parameter determines the maximum number of cursors per user. Before you start the Oracle historian the first time, increase the value of the OPEN CURSORS parameter to 200 or above. This setting is in the **INIT.ORA** file and has a valid range of 5 to 255. For instructions for increasing the value of OPEN_CURSORS, refer to the *Oracle RDBMS Database Administrator Guide*.

A setting of 200 cursors may not be high enough for extremely large applications. When this setting is not high enough, the following message is written to the directory defined by the environment variables in the log file **ohmmddyy.log**, where **oh** is the identifier for the Oracle historian and **mmddyy** represents the date.

FLAPP/FLNAME/FLDOMAIN/FLUSER/log:
ORA-01000: maximum open cursors exceeded

Increase the value of the OPEN CURSORS parameter to 255.

Data Types Supported by Oracle Database

For information on the supported data types, refer to the Oracle documentation. In older versions of FactoryLink, the format of the FactoryLink supported date data type had to be a string in the format of *yyyymmddhhmmss*. The tag had to be defined as a message with a minimum default length of 14 bytes to retrieve the date data type into a tag. If a tag was to insert or update a database row with the date data type, then the tag also had to be a message data type using previous format. FactoryLink now supports a direct conversion to and from a long analog tag (such as SECTIME) that equates to the elapsed seconds since January 1, 1980 to the date data type.

Historian Mailbox Information Table

Historians process data requests queued to a mailbox. A single historian can service multiple mailboxes; however, there is no requirement to create multiple mailboxes for an instance of the historian. There is also no advantage in using multiple mailboxes from an application behavior or performance standpoint. It is generally more efficient to use a single mailbox for each historian instance, regardless of the number of clients it services, than to have multiple mailboxes. However, some developers may choose to use different mailboxes for purposes related to application maintenance and readability. The mailboxes a historian services are defined in the Historian Mailbox Information table.

Accessing

In your server application, open **Historians > Historian for Oracle(R) > Historian Mailbox Information for Oracle(R).**

Field Description

Historian Mailbox	Mailbox name this historian services. This name must match the name defined in the task using historian to process data requests.

Create a separate mailbox for each task that submits data requests except for Database Logging and Trending, which can share a mailbox.

Valid Entry: tag name
Valid Data Type: mailbox

Historian Information Table

Database connection information is defined in the Historian Information table. Add at least one entry for each uniquely defined database to be accessed by this historian instance. Database connection information may be shared by multiple client tasks. Multiple entries for the same database connection are only necessary if separate control and monitoring of the connection status (and the tables that define) are required through the use of the Disable/Enable Connection, Connection Status, and Database Error tags. If the connection to a database via its defined alias closes for one task, it will close for all tasks. If separate entries are defined, then closing the connection for one alias does not affect the tasks using the other entries.

Accessing

In your server application, open **Historians > Historian for Oracle(R) > Historian Information for Oracle(R).**

Field Descriptions

Database Alias Name Unique name to represent a database connection. This must match the database name defined in the task using historian to process data requests.

Valid Entry: database connection name

Disable/ Enable Connection Name of a digital tag that enables or disables the connection. When this tag is set to one, the connection to the relational database defined in this entry is closed; when set to 0, the connection opens.

Note: Database aliases should not share connection tags. Sharing connection tags between database aliases can result in errors.

Valid Entry: tag name
Valid Data Type: digital

***Oracle User Name** Login name required to connect to the database. This name must be a valid Oracle account with connect, read/write, and create access to database tables. This name can either be a constant or a tag name.

If you enter a constant, precede the user name with a single quote.

The tag specified must be a message tag type if you enter a tag name. You must specify a login name in the tag **Default Value** field and a maximum length of 32 in the **Length** field.

Valid Entry: tag name or constant
Valid Data Type: message of up to 32 characters

*Oracle Password — Password required to connect to the database. This password can be either a constant or a tag name.

If you enter a constant, precede the password with a single quote.

The tag specified must be a message tag type if you enter a tag name. You must specify a password name in the tag **Default Value** field and a maximum length of 32 in the **Length** field.

Valid Entry: tag name or constant
Valid Data Type: message of up to 32 characters

*SQL *NET Connect String — SQL*Net connection string required to connect to the database. Leave this field blank if you want to use the default connection. This can either be a constant or a tag name.

If you enter a constant, precede the connection string with a single quote.

The tag specified must be a message tag type if you enter a tag name. You must specify the connection string in the tag **Default Value** field and a length in the **Length** field that accommodates the longest connection string you define for this tag. If the connection string exceeds the defined length, it is truncated and the connection is not made. Define 64 as the length for the best results.

Valid Entry: tag name or constant
Valid Data Type: message of up to 64 characters

Connection Status — Tag updated by the Historian that defines the state of this connection.

Note: Database aliases should not share status tags. Sharing status tags between database aliases can result in errors.

Valid Entry: tag name
Valid Data Type: analog

Database Error — Name of a tag to receive the error value passed from the database software.

The tag specified must be either long analog or message. It should correspond to the type of error the relational database sends. If it is a number use long analog; if it is text use message.

Valid Entry: tag name
Valid Data Type: longana, message

Connecting Historian to an Oracle Database

Two key areas must be set for your Oracle historian to pass data requests to your Oracle RDBMS.

Granting FactoryLink Account Oracle Access

You must grant access to the user account for the historian to exchange data with an Oracle database, meaning the username and password, specified in the Historian Information table. For the instructions on how to create an Oracle user account, refer to the *Oracle System Administration Guide*.

The Oracle user account must have system privileges to connect to a database and delete, update, insert, and select rows from a database table. Additionally, if the FactoryLink application requires, this account may also need to create table and index privileges.

Setting Connection Strings

You must set the connection strings. This section provides the syntax to connect to SQL*Net V1 and V2 clients. Refer to the SQL*Net documentation set for your Oracle server running on your server host before you define a connection string to any platform.

*SQL*Net V1 Syntax*

@*prefix*:*host_name*:*system_ID*

where

@	Marks the start of the connection string
:	Is a field delimiter
prefix	Represents the network transport
host_name	Is the server host
system_ID	Is the Oracle system ID

This is an example connection string for the SQL*Net TCP/IP network protocol to a UNIX server:

@T:FLORASRV:B

where

@	Marks the start of the connection string
T	Represents the TCP/IP network transport

FLORASRV	Is the server host
B	Is the Oracle System ID

*SQL*Net V2 Syntax*

@alias

where

@	Marks the start of the connection string.
alias	Is an alias name defined in an SQL*Net V2 configuration file

This example is valid across all platforms.

@FLORACLE	where
FLORACLE	Is an alias configured in an SQL*Net V2 configuration file

SYBASE HISTORIAN

This section provides information needed to configure the Sybase Historian.

Considerations

By default, 25 is the maximum number of Sybase connections allowed per process; however, you can increase this maximum, which is limited only by system resources, by changing the value set by the environment variable MAXDBPROCS. The Historian checks MAXDBPROCS against the actual number of Sybase connections per process; therefore, if you want to use more than 25 Sybase connections per process, set the environment variable MAXDBPROCS to that value or greater.

Supported Data Types

For information on the data types supported by the Sybase database, refer to the Sybase documentation. The Sybase Historian task does not support text and image data types.

In older versions of FactoryLink, the format of the FactoryLink supported date data type had to be a string in the format of *yyyymmddhhmmss*. The tag had to be defined as a message with a minimum default length of 14 bytes to retrieve the date data type into a tag. If a tag was to insert or update a database row with the date data type, then the tag also had to be a message data type using previous format. FactoryLink now supports a direct conversion to and from a long analog tag (such as SECTIME) that equates to the elapsed seconds since January 1, 1980 to the date data type.

Historian Mailbox Information Table for Sybase

Historians process data requests queued to a mailbox. A single Historian can service multiple mailboxes; however, there is no requirement to create multiple mailboxes for an instance of the historian. There is also no advantage in using multiple mailboxes from an application behavior or performance standpoint. It is generally more efficient to use a single mailbox for each historian instance, regardless of the number of clients it services, than to have multiple mailboxes. However, some developers may choose to use different mailboxes for purposes related to application maintenance and readability. The mailboxes a Historian services are defined in the Historian Mailbox Information table.

Accessing

In your server application, open **Historians > Historian for Sybase(R) > Historian Mailbox Information for Sybase(R)**.

Field Description

Historian Mailbox — Mailbox name this Historian services. This name must match the name defined in the task using Historian to process data requests.

Valid Entry: tag name
Valid Data Type: mailbox

Historian Information Table for Sybase

Database connection information is defined in the Historian Information table. Add at least one entry for each uniquely defined database to be accessed by this historian instance. Database connection information may be shared by multiple client tasks. Multiple entries for the same database connection are only necessary if separate control and monitoring of the connection status (and the tables that define) are required through the use of the Disable/Enable Connection, Connection Status, and Database Error tags. If the connection to a database via its defined alias closes for one task, it will close for all tasks. If separate entries are defined, then closing the connection for one alias does not affect the tasks using the other entries.

Accessing

In your server application, open **Historians > Historian for Sybase(R) > Historian Information for Sybase(R).**

Field Descriptions

Database Alias Name — Unique name to represent a database connection. This must match the database name defined in the task using Historian to process data requests.

Valid Entry: database connection name

Disable/Enable Connection — Name of a digital tag that enables or disables the connection. When this tag is set to one, the connection to the relational database defined in this entry is closed; when set to 0, the connection opens.

Note: Database aliases should not share connection tags. Sharing connection tags between database aliases can result in errors.

Valid Entry: tag name
Valid Data Type: digital

*Server Name — Server name you want to connect to. The server name is the alias given to the server in the Sybase Interfaces files. This name can either be a tag name or a constant preceded by a single quote.

Valid Entry: tag name or string constant of up to 32 characters
Valid Data Type: message

*Database Name
Sybase database to use. This database must exist before restarting the Historian at run time. This name can either be a tag name or a constant preceded by a single quote.

Valid Entry: tag name or string constant of up to 32 characters
Valid Data Type: message

*Server User Name
Login name required to connect to the database. This name must be a valid Sybase account with connect, read/write, and create access to database tables. This name can either be a tag name or a constant preceded by a single quote.

Valid Entry: tag name or string constant of up to 32 characters
Valid Data Type: message

*Server Password
Password required to connect to the database. This password can either be a tag name or a constant preceded by a single quote.

Valid Entry: tag name or string constant of up to 32 characters
Valid Data Type: message

Connection Status
Tag updated by the Historian that defines the state of this connection.

Note: Database aliases should not share status tags. Sharing status tags between database aliases can result in errors.

Valid Entry: tag name
Valid Data Type: analog

Database Error
Tag to receive the error value passed from the database software.

The tag specified must be one of the following types: long analog or message. It should correspond to the type of error the relational database sends. If it is a number use long analog; if it is text use message.

The database error tag is updated only when a fatal error is defined in the FLHST.INI file or if a database open connect call fails.

Valid Entry: tag name
Valid Data Type: longana or message

User Access to Server and Database

You must enable the Historian to access the Sybase database before the Historian for Sybase can exchange data with a Sybase database.

Perform these steps to establish this connection:

1. Install the Sybase SQL server.
2. Modify the interfaces file.
3. Create Sybase databases. Create all Sybase databases for FactoryLink to use before you start up the Historian.
4. Add login names for server users.
5. Add server users to the database.
6. Grant permission to the FactoryLink account to use **CREATE PROC** and **CREATE TABLE** commands.

   For instructions on how you complete these steps, refer to the *Sybase System Administration Guide* and *Sybase Commands Reference*.

Setting Sybase Interfaces File

All FactoryLink users must be able to read all interface files, which must reside in the Sybase home directory. The Sybase home directory is the directory where Sybase is installed. This can be a Sybase product that allows you to connect remotely.

Use the following guidelines to modify the interfaces file:

1. Verify the interface file location is correct, based on:
   - If Sybase SQL server and FactoryLink run on the same machine, the interfaces file resides in the Sybase home directory.
   - If the Sybase SQL server and FactoryLink run on a different machines, create the interfaces file in the Sybase home directory.
2. Add one entry for each SQL server to the interfaces file when using more than one Sybase SQL server.

Granting Users Access to Create Commands

Authorized FactoryLink users of the Sybase server need permission to use the **CREATE PROC** and **CREATE TABLE** commands.

Perform the following steps to grant this permission:

1. Log into the Sybase SQL server as the System Administrator.
2. At the Sybase SQL server prompt, enter

   use database

   go

   where

   database Is the name of the Sybase database FactoryLink uses.
3. At the Sybase SQL server prompt, enter

   grant CREATE PROC, CREATE TABLE to username

   go

   where

   username Is the name of the user accessing the Sybase SQL server.

dBASE IV Historian

The dBASE IV Historian is file-based. For large applications, it is recommended that you use a standard multi-tier database, such as SQL Server, Oracle, or Sybase. For smaller applications or applications with minimal logging requirements, the dBase IV Historian may be adequate.

This section describes how to configure connection information for dBASE IV Historian, which includes defining dBASE IV Mailboxes and defining dBASE IV Connection Information.

dBASE IV Historian Considerations

The Historian supports access to network drives located on a file server. Data types supported by dBASE IV Historian are Character, Date, Decimal, Float, Integer, and Smallint.

In previous versions of FactoryLink, the format of the date data type had to be a string in the format of *yyyymmddhhmmss*. The tag had to be defined as a message with a minimum default length of 14 bytes to retrieve the date data type into a tag. If a tag was to insert or update a database row with the date data type, then the tag also had to be a message data type using previous format. A direct conversion to and from a long analog tag (such as SECTIME) that equates to the elapsed seconds since January 1, 1980 to the date data type is now supported.

Historian Mailbox Information Table

Historians process data requests queued to a mailbox. A single Historian can service multiple mailboxes, however, there is no requirement to create multiple mailboxes for an instance of the historian. There is also no advantage in using multiple mailboxes from an application behavior or performance standpoint. It is generally more efficient to use a single mailbox for each historian instance, regardless of the number of clients it services, than to have multiple mailboxes. However, some developers may choose to use different mailboxes for purposes related to application maintenance and readability. The mailboxes a Historian services are defined in the Historian Mailbox Information table.

Accessing

In your server application, open **Historians > Historian for dBASE IV(R) > Historian Information for dBASE IV.**

Field Description

Historian Mailbox — Mailbox this Historian services. This name must match the name defined in the task using Historian to process data requests.

Valid Entry: mailbox name

Historian Information Table

Database connection information is defined in the Historian Information table. Add at least one entry for each uniquely defined database to be accessed by this historian instance. Database connection information may be shared by multiple client tasks. Multiple entries for the same database connection are only necessary if separate control and monitoring of the connection status (and the tables that define) are required through the use of the Disable/Enable Connection, Connection Status, and Database Error tags. If the connection to a database via its defined alias closes for one task, it will close for all tasks. If separate entries are defined, then closing the connection for one alias does not affect the tasks using the other entries.

Accessing

In your server application, open **Historians > Historian for dBASE IV(R) > Historian Information for dBASE IV.**

Field Descriptions

Database Alias Name — Unique name to represent a database connection.

Valid Entry: database connection name

Disable/ Enable Connection — Digital tag name that enables or disables the connection. When this tag is set to 1, the connection to the relational database defined in this entry is closed; when set to 0, the connection opens.

Valid Entry: tag name
Valid Data Type: digital

Note: Database aliases should not share connection tags. Sharing connection tags between database aliases can result in errors.

Database Directory — Directory path name where the database resides. This directory must exist before starting this Historian at run time. The path can be either a constant or a tag name.

If you specify a constant, precede the constant by a single quote.

If you specify a tag name, the tag specified must be of type message. You must specify the directory path where the database resides in the tag Default Value field and a length in the Length field that accommodates the longest directory path you might define for this tag. If the directory path exceeds the defined length, it is truncated and the connection cannot be made. Define 66 as the length for the best results.

Valid Entry: tag name or string constant
Valid Data Type: message or string constant of up to 66 characters

Connection Status — Name of an analog tag to receive the connection status between the Historian and the relational database. Refer to “Database Disconnects and Reconnects” on page 280 for a description of the status values.

Valid Entry: tag name
Valid Data Type: analog

Note: Database aliases should not share status tags. Sharing status tags between database aliases can result in errors.

Database Error — Name of a tag to receive the error value passed from the database software. The Database Error tag is updated only when a fatal error is defined in the flhst.ini file or if a database open connect call fails.

The tag should correspond to the type of error the relational database sends.

Valid Entry: tag name

Maintaining dBASE IV Database Files

Use the interactive BH_SQL and DBCHK utilities to maintain FactoryLink dBASE IV relational databases. Users of dBASE IV files must have both read and write access. For more information on how to use BH_SQL and DBCHK, see the *Utilities Guide*.

Reserved Words

The Historian uses the following reserved words with dBASE IV. Do not use these keywords when defining table or column names.

ALL	DECIMAL	INTEGER	SET
ALTER	DELETE	INTERSECT	SMALLINT
AND	DESC	INTO	SOME
ANY	DESCENDING	IS	SUM
ASC	DISTINCT	LIKE	SYNONYM
ASCENDING	DROP	MAX	TABLE
AVG	EXISTS	MIN	UNION
BETWEEN	FLOAT	MINUS	UNIQUE
BY	FOR	NOT	UPDATE
CHAR	FROM	NULL	VALUES
CHARACTER	GROUP	NUMBER	VARCHAR
COUNT	HAVING	NUMERIC	VIEW
CREATE	IN	ON	WHERE
CURRENT	INDEX	OR	
DATE	INSERT	ORDER	
DEC	INT	SELECT	

DATABASE DISCONNECTS AND RECONNECTS

The state of the connection is maintained in a Connection Status tag. Possible states are described in the following table. The state determines the acceptable actions you can take to any table in the database accessible by FactoryLink.

Connection State	Description
0 = Database Connection Inactive	Indicates historian is not connected to the database. At startup, historian writes a zero to the status tag for each database. You can perform maintenance on the database outside of FactoryLink while the connection is in this state.
10 = Normal connection	Indicates historian is connected to database. SQL data requests from FactoryLink tasks can be transmitted to and from the database.
20 = Connecting to database	Request for a connection in progress.
30 = Disconnecting from database	Connection is in the process of being disabled. When the connection is disabled, the status returns to zero. You do not want to maintain the database until the connection status is zero.
110 = Fatal Error Condition/Lost Connection	Indicates the connection failed, resulting in a fatal error. If you receive a fatal error status, you can find out about what caused the error from the contents of the database error tag.

Disconnects from a relational database can occur for either of the following reasons:

- They are scheduled to occur at predefined times.
- A fatal error forces an unscheduled disconnect.

Scheduled Disconnects

You can schedule disconnects on a periodic basis to perform maintenance on relational database tables FactoryLink accesses. When you schedule a disconnect, you also have the option of changing the connection from the database you are disabling to another database so data requests can continue to be processed.

Configuring Scheduled Disconnects

You must configure a FactoryLink task to set the disable/enable connection tag defined on the Historian Configuration table to 1 to initiate a disconnect. Once a connection is disabled, the historian returns a HSDISABLED error code to the requesting tasks. All data is lost during the period of disconnect.

You must configure a FactoryLink task to write the connection strings required to connect to the new database to the tags that define the connection you want to change. These tags are specified in the Historian Configuration tables. Then write 0 to the disable/enable connection tag defined on the Historian Configuration tables.

Unscheduled disconnects can occur because of fatal errors. Historians detect fatal error conditions returned either by the RDBMS server or the network client software. The historian tasks consider an error condition to be fatal when an error code generated by a database server is found in the Fatal Error Codes list you defined in the **FLINK/bin/flhst.ini** file. For more information on how to define these codes, see “Setting Run-Time Fatal Error Code Values” on page 281.

Database Reconnect

Database reconnect provides the ability to reconnect to a database when the connection has been lost. Historian reconnect is only valid when the task is running. The historian information may not get updated after the reconnect if a screen is open when the database is disconnected and reconnected. If this occurs, exit and reenter the screen to refresh historian updating. Reconnect does not work if the historian is brought down and then brought back up.

Setting Run-Time Fatal Error Code Values

You must define the range of numeric codes that qualify as fatal for Historian to detect fatal run-time errors. If you are using ODBC, you must also define the data sources and servers supported by each operating system platform.

Fatal run-time error codes and ODBC support information are configured in the **FLINK/bin/flhst.ini** file. This file contains a section for each historian and one or more ODBC database server names.

Alphanumeric Error Codes in ODBC

Any ODBC alphanumeric error code must be surrounded by quotation marks. The alphanumeric error code consists of two parts. The first part is the ODBC “state” string. The second part (enclosed in parentheses) is the native error produced by the database.

The following examples illustrate the correct syntax.

List of alphanumeric error codes	**FatalErrorCodes=”08S01(0)”, “08S01(11)”,”08S01(14)”**
State string only	**FatalErrorCodes=”08S01”**

Range of alphanumeric error codes	**FatalErrorCodes=”08S01(0)” TO ”08S01(14)”**

The **FLINK/bin/flhst.ini** configuration file is divided into sections. Each section represents a different ODBC data source name for ODBC support information or a different historian for definition of fatal error codes.

Following is an explanation of these entries:

#	A pound sign in column 1 identifies a comment that the historian task ignores.
[taskname]	Name entered in the System Configuration table. For ODBC, it is a data source.
IllegalDuplicateKey	Single value valid only for an ODBC data source. Other historians ignore this.
FatalErrorCodes	Single value or a range of values considered fatal run-time errors to the historian. The Fatal Error Codes can be a single value or a single range of values. It can also be a list where each value or a range is delimited by a comma.
SQLDescRequired = True	Reserved words ASCENDING and DESCENDING are shortened to ASC and DESC for proper syntax usage by the database server.

Use any text editor to edit this file.

Following is a sample of the contents for this file.

Sample **flhst.ini** File

ODBC driver & Data source

```
# Microsoft Access ODBC Driver
[ACCESS]
IllegalDuplicateKey=-1605
```

One fatal error code

```
FatalErrorCodes="23000(-1608)"

# Microsoft SQL Server ODBC Driver
[Microsoft SQL Server]
IllegalDuplicateKey=2601
```

Undefined entry for fatal codes.

```
FatalErrorCodes=

# Oracle Version 7 ODBC driver
```

Historian Task Name

```
[Oracle7]
IllegalDuplicateKey=1
```

List of fatal codes

```
FatalErrorCodes=6000 to 6429, 6600, 6610, 7000 to 7100

# ORACLE Version 7 FactoryLink Historian
[OR7_HIST]
```

Range of fatal error codes

```
FatalErrorCodes=6000 to 6429
```

Note: The error code values listed in the Fatal Error Codes example are not actual error codes. For the actual codes, refer to the RDBMS user's manual.

Handling Fatal Errors

If an unscheduled disconnect occurs because of a fatal error, the information about what caused the fatal error is written to the database error tag. The format of the error message depends on the error tag data type.

If the error tag data type is message, the error message is written in the following format:

taskname:err_msg

where

taskname Is the historian task name that initiated the error condition.

err_msg Is the text from the relational database server.

If the error tag data type is a long analog, the tag contains the database-dependent error code number.

Every time the relational database server returns an error code to the historian, the historian tests this code against the range defined in the **flhst.ini** file.

When a historian determines an error is fatal, it sets the connection status tag to 110. What happens next depends on how the FactoryLink application is configured to handle fatal errors. Your FactoryLink application can reconnect to the database after an error has been resolved.

One approach is to have the FactoryLink application set the disable/enable connection tag to 1 to disable the connection to the database causing the error, then attempt to reconnect by setting the disable/enable connection tag to 0.

PROGRAM ARGUMENTS

Database Type	Argument	Description
Common	-DATE<f>	Set date or time format. <f> = format string of the following tokens: Token Description year 4-digit year yr 2-digit year mo Month of year dy Day of month hr Hour (24-hour clock) mi Minute sc Second These tokens must be separated by a pipe (|), a colon (:), or a blank space.
	-E	Echo database errors to Run-Time Manager.
	-i<#>	Maximum cursor inactivity time, in minutes, before the historian task closes the SQL cursor. The default is 15 minutes. (# = 15 to 300 minutes)
	-I<#>	Set a maximum time in minutes of connection inactivity before the historian task closes the SQL connection. The default is to never close the connection. (# = 15 to 4320 minutes)
	-L<#> or -l<#>	Enables logging of debug information to a log file. By default logging is enabled. The historian always logs errors to the log file. The transaction is logged if a logging level is specified. Level 1 logs OPEN/CLOSE transactions for SQL connections and cursors. Level 2 logs the EXECUTE and FETCH transactions as well as level 1 transactions. Level 3 logs level 1 and 2 transactions and also logs the data supplied by the EXECUTE transaction. Level 3 may generate a tremendous amount of data. Usually these levels are used for debugging a FactoryLink application and should be removed for normal operation.

Database Type	Argument	Description
Common (continued)	-M<m>	Set execute mode. (m = BATCH|NOBATCH)
	-S	Display database requests in Run-Time Manager output window.
	-TIME<f>	See -DATE<f> above.
	-U<#>	Set unique ID; for use with Multiple Instance Historian. (# = 0 to 9)
	-V<#>	Set verbose level. (# = 0 to 4)
	-X or -x	Adds extra lines to the log file giving times when historian sent and received messages. The message ID sequences these lines for better cross-referencing with similar lines in the historian task log file.
dBaseIV	-C	Checks database files for possible corruption before opening the dBASE IV file. This check is done only if the dBASE IV Historian has this argument and the computer is improperly shut down, or if the dBASE IV Historian task aborts. In either situation, some dBASE IV files will not have closed properly. Once the dBASE IV historian task is restarted, the dBASE IV historian detects the improper event and does not allow the dBASE IV files to open until the BH_SQL task verifies them.
	-F<#>	Sets maximum number of file handles that the dBASE IV historian can open concurrently. (# = 40 to 200). A low value my cause significant performance problems for applications that have large numbers of tables/databases. This is due to the constant closing and opening of files. Increase this value if performance problems occur.
	-M	Allows the dBASE IV historian to be multiuser. Every update, insert, and delete causes the dBASE IV file to be flushed to disk. This allows other third-party software to view the data as it is added or modified by the dBASE IV historian.
	-dbase	Generate dBaseIV-compatible database files (with extension = .dbf).

Database Type	Argument	Description
Oracle	-otimeout<#>	Set timeout to wait on a blocked call. Default is 30. (# = seconds)
	-osleep<#>	Set sleep time to retry a blocked call. Default is 50. (# = milliseconds)
	-omode<m>	Set Oracle DB mode. (m = NBL|DEF). For example, **-omode=NBL** enables nonblocking mode. Default is blocking. When using the nonblocking mode, the enable/disable tag must be cycled for any alias that experiences a timeout in order to clear out pending results from the request on the server. It is possible that after the historian timed out, the server actually completed the request successfully. In this case, the connection to the server may be left in a state that requires the result to be passed back to the historian. This will prevent the execution of any new SQL statements or receiving erroneous returned data if the same SQL statement that timed out is executed again.

TROUBLESHOOTING

You can set two types of files to record Historian operations: a Historian log file and a Historian trace file. These two files will best help you and your support representative troubleshoot your FactoryLink application.

Historian log files are accessible on disk for seven days. After seven days, old log files are deleted. At the start of each new day, the previous day log file closes and a new one opens.

Log files reside in the **FLAPP/FLNAME/SHARED/FLUSER/log** directory. The name of a log file follows the format of

PRMMDDYY.**log**

where

PR	Identifies the Historian name.
MM, DD, YY	Are two-digit numerals for the month, day, year the log file was created.

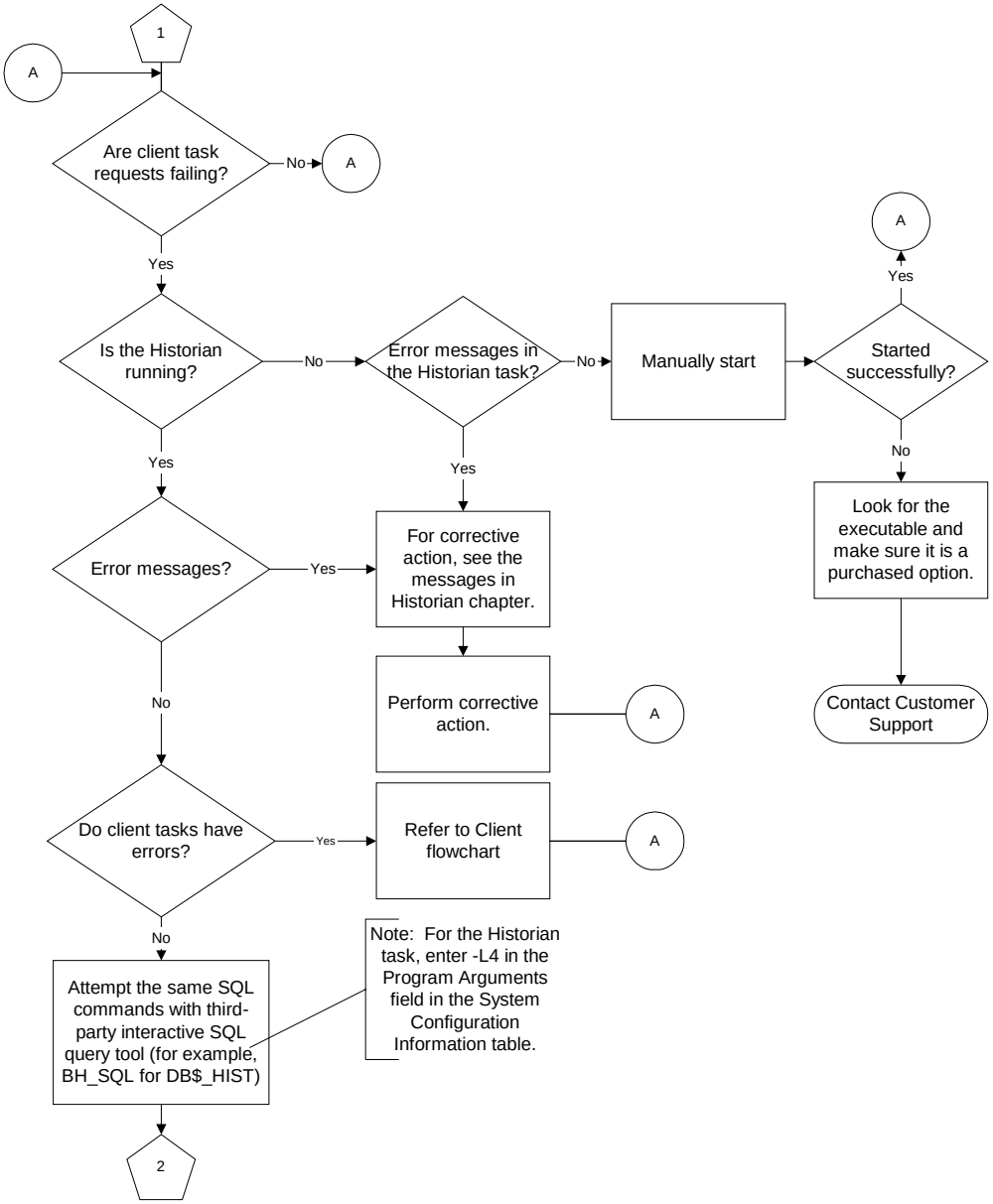
The following table lists the prefix and sample log file names for each Historian.

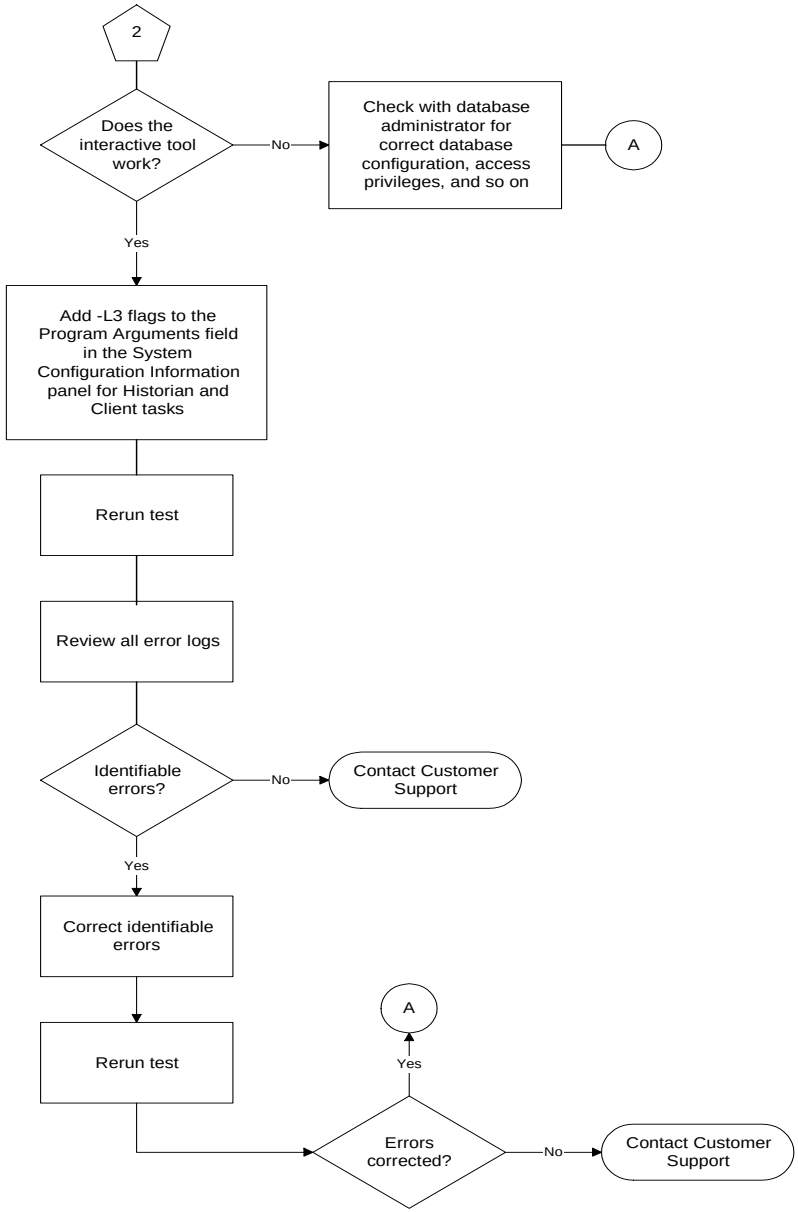
Historian	Prefix	Log file - Sample name
Oracle	oh	oh010195.log
DB IV	d4	d4010195.log
ODBC	od	od010195.log
Sybase	sy	isy010195.log

Entries continue to append to each **.log** file. Consequently, these files can grow and take up large amounts of disk space. Delete the file contents or the file itself. Remove the Program Arguments on the System Configuration Information table to stop logging for a Historian.

> **Note:** Be aware that, when using multiple historians, some or all transactions for the various clients are synchronous while others are asynchronous. If a client executes a synchronous transaction with one historian and it does not respond for whatever reason, the client must wait until the timeout period for that transaction to elapse before it can process any other triggered transactions for any of the historians.

Use the following flowchart for help in troubleshooting the logging configuration.


1
A
Are client task requests failing?
No
A
Yes
Is the Historian running?
No
Error messages in the Historian task?
No
Manually start
A
Started successfully?
Yes
No
Look for the executable and make sure it is a purchased option.
Contact Customer Support
Yes
Yes
Error messages?
Yes
For corrective action, see the messages in Historian chapter.
Perform corrective action.
A
No
Do client tasks have errors?
Yes
Refer to Client flowchart
A
No
Attempt the same SQL commands with third-party interactive SQL query tool (for example, BH_SQL for DB$_HIST)
Note: For the Historian task, enter -L4 in the Program Arguments field in the System Configuration Information table.
2


2
Does the interactive tool work?
No
Check with database administrator for correct database configuration, access privileges, and so on
A
Yes
Add -L3 flags to the Program Arguments field in the System Configuration Information panel for Historian and Client tasks
Rerun test
Review all error logs
Identifiable errors?
No
Contact Customer Support
Yes
Correct identifiable errors
Rerun test
A
Yes
Errors corrected?
No
Contact Customer Support

ODBC Driver

Display the **Data Source Setup** dialog and perform the following steps to troubleshoot the ODBC driver:

1. Highlight your data source and click Options.
2. Click the **Trace ODBC Calls** check box to enable tracing.

   By default, the Stop Tracing Automatically check box is enabled, which sets tracing to stop automatically upon a disconnect from the data source. Make note of the trace file location—**SQL.log**. You can change this location.
3. Click **Select File**.
4. Define a new location. Click **OK**, and then click **Close**.

HISTORIAN MESSAGES

Messages may come from FactoryLink, a database driver, or data source. Messages communicate a status or a condition that may or may not require an action from you.

Run-Time Messages

FactoryLink Historians generally do not write error messages generated from a data source to the Task Status tag. The Historian reports all database errors to its log file and returns a status code to the Historian client tasks, such as Database Browser for every Historian operation.

Errors and messages may display as your FactoryLink application runs. FactoryLink sends a code or message to the Run-Time Manager screen for display whenever an error occurs in a Historian or a Historian-client task. You can also define an output text object to display codes and messages on a graphics screen.

FactoryLink also sends a longer, more descriptive message to the log file when the log file Program Argument is set. The Task Status tag is located on the System Configuration Information table. The data type you assign to this tag for a Historian and any FactoryLink task determines the type of codes written to this tag. You can assign these data types:

- Digital data type reports these two codes

  0—indicates the requested operation successfully completed

  1—indicates an error occurred

- Analog, longana, or float data type

  A Historian may generate the following codes on the Run-Time Manager screen. These status codes are returned to all tasks requesting data from a Historian. For a list of the Database Browser status codes and status messages, see “Database Browser” on page 73.

Code	Error	Cause	Action
-1	A FactoryLink error occurred.	A read or a write to a mailbox occurred for either a Historian-client or a Historian task.	Contact your technical support representative.
0	Successful operation with no return data.	Historian operation was successful.	No action required.
1	Successful operation with return data.	Historian operation was successful.	No action required.
2	Unsolicited data sent to a Historian client task.	An unsolicited fetch was performed for a Historian-client task.	No action required.
3	Maximum OPEN connections exceeded.	A Historian-client task tried to access data from more than 10 unique database aliases.	Reference fewer than 10 unique database aliases for a Historian-client task.
4	A dynamic memory allocation operation failed.	An internal error occurred in a Historian-client task or in a Historian task.	Contact your technical support representative.
5	Historian-client task referenced an invalid Historian connection handle.	An internal error occurred within a Historian-client task.	Contact your technical support representative.
6	Failed to prepare a Historian statement handle.	An internal error occurred within a Historian-client task.	Contact your technical support representative.
7	Invalid stmtid returned from Historian	1) Historian shut down before Browser or another Historian-client task shut down.	1) Shut down all Historian client tasks running on the system. Then, shut down the Historian. Then, start up the Historian and then Browser and all other Historian-client tasks running on the system.

Code	Error	Cause	Action
8	Last row fetched, or no rows affected by SQL operation.	Historian task operation affected no rows for an UPDATE, DELETE operation or a FETCH for a SELECT operation reached the end of the result table. In addition, for Database Browser an attempt to move past the end or before the beginning of the result table generates this error code.	No action required. An informational message only. An appropriate action is a FactoryLink application logic decision.
9	A Historian database alias is invalid.	A Historian-client task referenced a Historian Database Alias that does not match any of the entries defined in the Historian Information table.	Ensure all database aliases referenced by the Historian client task exist in the Historian Information table.
10	An operation exceeded its maximum timeout period.	1. The mailbox specified as a Historian mailbox is not listed in the Historian mailbox table. 2. SQL operation is taking too long to perform. 3. Historian task is busy with other operations and is unable to service new Historian-client requests.	1. Occurs in an OPEN connection call. Ensure mailbox exists in the Historian mailbox table. 2. Set the program arguments to -L2 -X on the Historian to figure out how long the database server is taking to perform SQL operation. Either change the SQL statement to perform less work or have the database server do this operation instead of the Historian task. 3. Slow down the amount of work the Historian is doing by either ensuring the SQL operations are not time-consuming or slow down the Database Logger to trigger less frequently.
11	A Historian-client task passed an invalid Historian connection handle to a Historian.	The Historian task was terminated before other Historian-client tasks.	Shut down all Historian-client tasks running on the system and then the Historian. Restart Historian followed by all other Historian-client tasks.

Code	Error	Cause	Action
12	A Historian-client task passed an invalid Historian statement handle to a Historian.	1. The Historian task was terminated before other Historian-client tasks.	Shut down all Historian-client tasks running on the system and then the Historian. Restart Historian followed by all other Historian-client tasks.
14	Historian-client task operation failed because the Historian disabled the connection to the database.	The FactoryLink application has disabled the connection to the database by setting the Historian Disable/Enable tag.	Enable the Historian Disable/Enable tag to perform this operation.
16	An SQL operation is not supported.	A Historian-client task has used an SQL feature not supported by the Historian.	If using dBASE IV, consult the chapter on SQL limitations for the dBASE IV Historian. If other Historian, see the Historian log file for the SQL statement and then consult appropriate user's manual.
17	Historian database error.	This is a general error code for the Historian. The error text string may contain more information. This code is accompanied by various descriptive messages.	Read the accompanying message displayed on the Run-Time Manager screen or in the .LOG file and attempt to correct the problem based on that message's instructions.
18	Unknown function request sent to Historian	An internal error occurred within a Historian-client task.	Contact your technical support representative.
19	Tried to insert a duplicate row.	INSERT operation caused a UNIQUE constraint error on the database table.	No action required.
20	Table exists, tried to create a table that already exists.	CREATE TABLE operation failed because the table name already exists in the database.	No action required.
22	Table does not exist.	You tried to access a table that does not exist in database.	No action required.
23	Field does not exist.	You tried to access a field that does not exist in database table.	No action required.

Startup Messages

The following messages may display on the Run-Time Manager screen if an error occurs with Historian at startup. See the Historian's .log file for the complete message.

Error Message	Cause and Action
Bad CT record size in CT archive *filename*	Cause: The archive file *HIST.CT is corrupt. Action: Delete the archive file *HIST.CT. Restart the application to rebuild the archive file.
Bad CT type in CT archive *filename*	Cause: The archive file *HIST.CT is corrupt. Action: Delete the archive file *HIST.CT. Restart the application to rebuild the archive file.
Bad hist. mailbox in CT archive *filename*	Cause: The file *HIST.CT contains an invalid tag name for a mailbox tag. Action: Delete the invalid tag name from the Historian Mailbox Information table and enter a valid tag name.
Can't open CT archive *filename*	Cause: Either the archive file ***HIST.CT** does not exist or is corrupt. Action: Delete it if the file exists. Restart the application to rebuild the archive file.
Can't open .LOG file	Cause: The Database Logging task can not open a .LOG file. Action: Ensure the **FLAPP/LOG** directory exists.
Can't read record in CT archive *filename*	Cause: The archive file ***HIST.CT** is corrupt. Action: Delete the archive file ***HIST.CT**. Restart the application to rebuild the archive file.
Duplicate mailboxes in CT archive *filename*	Cause: The Historian Mailbox Information table contains duplicate mailbox tags. Action: Delete the duplicate entry in the Historian Mailbox Information table.
Invalid # *number* of CT types in CT archive *name*	Cause: The archive file ***HIST.CT** is corrupt. Action: Delete the archive file ***HIST.CT**. Restart the application to rebuild the archive file.
No CTs in CT archive *filename*	Cause: The Historian Table is not configured. Action: Complete the Historian table.

Error Message	Cause and Action
No dbases defined in CT archive *filename*	Cause: The Historian Information table is not configured. Action: Complete the Historian table.
No mailboxes in CT archive *filename*	Cause: The Historian Mailbox Information table is not configured. Action: Complete the Historian Mailbox Information table.
Out of RAM	Cause: Not enough RAM to run this task. Action: Restart the task. If it fails again, allocate more RAM for this task or install more RAM.
SQLERROR: Can't function for task *task_name* due to error: *error*	Cause: An SQL error occurred during the execution of an SQL statement. Action: Refer to the user manual for the database in use for instructions for correcting this error.
SQLERROR: Can't function serial id with SQL statement: *sql_statement* for task *task_name* due to error: *error*	Cause: An SQL error occurred during the execution of an SQL statement for a FactoryLink task. Action: Refer to the user manual for the database in use for instructions for correcting this error.

Error Messages Recorded in Historian Log Files

The following messages may display in a Historian .log file.

Error Message	Cause and Action
SQLERROR: Can't sql_function SQL statement *sql_statement* for task *task_name* due to error: *error*	Cause: An SQL error occurred during the execution of an SQL statement for a FactoryLink task. Action: Refer to the documentation for the database in use for instructions for correcting this error.
SQLERROR: Can't function unique index: *unique_index* for task *task_name* due to error: error	Cause: An SQL error occurred during the execution of an SQL statement for a FactoryLink task. Action: Refer to the documentation for the database in use for instructions for correcting this error.
Task initialization failed	Cause: The Historian task is installed incorrectly. Action: Reinstall FactoryLink.

ODBC Driver and Data Source Messages

This section explains messages that can occur during the running of the ODBC Historian. ODBC messages can come from the following:

- Driver manager
- Database manager
- ODBC driver
- The driver manager is a DLL that establishes connections with drivers, submits requests to drivers, and returns results to applications. An error that occurs in the driver manager has the following format:

  [*vendor*][*ODBC DLL*]*message*

  where

  vendor Is Microsoft.

- An error that occurs in the database system includes the database name. For example, you might get the following message from an Oracle data source:

  [Microsoft][ODBC *XXX driver*][*XXX*]*message*

  If you get this type of error, you have attempted to do something incorrectly with the database system. Check your system documentation for more information or consult your database administrator.

- An error that occurs in an ODBC driver has the following format:

  [Microsoft][ODBC *XXX driver*]*message*

Chapter 14. . . .

Math and Logic

The Math and Logic task performs mathematical and logical operations on tags in the Real-Time Database. The results are stored in tags for use by other tasks. Two modes (interpreted and compiled) are available, permitting users to optimize applications for maximum performance. The Math and Logic functions include the following types of operations:

- Arithmetic
- Logical
- Exponential
- String Manipulation
- User-defined C Routines (compiled mode)
- Relational
- Trigonometric
- Logarithmic
- If-Then-Else or While Functions

The Math and Logic task:

- Uses a structured, BASIC-like syntax
- Executes in the background with no operator intervention at run time
- Supports block structures, looping constructs, and callable mathematical functions
- Supports the following data types: digital, analog, long analog, floating point, and message
- Supports multi-dimensional arrays

MODES

Math and Logic runs in one of two modes: Interpreted Math and Logic (IML) and Compiled Math and Logic (CML). It is possible to run both modes at the same time, but the application designer must be sure any procedures called from a compiled procedure are also configured as compiled.

The FactoryLink designer must determine the type of mode to use. A comparison of the two modes is in the following table. Most applications written in the Interpreted Mode function can be used with limited or no modifications under the Compiled Mode. If an application running in Interpreted Mode uses any reserved words as variables or procedure names, these must be modified before they can be used in CML. These words include any reserved by the compiler you are using and those reserved by FactoryLink.

Table 14-1 Comparison of IML and CML Modes

IML	CML
Interpreted mode (standard module)	Compiled mode (optional module for purchase)
Most appropriate when only mathematical functions are performed or the application is relatively small Excellent for application prototyping and logic debugging	Most appropriate when you want: • Faster startup and execution for improved performance • Better handling of complex, multiple conditional statements • Ability to call C functions or to insert C code into procedures • Availability of extensive flow control for the application
Executes arithmetic operations fairly quickly	Adds more functionality as an in-line function to pass simple parameters and as an embedded C code block to pass complicated parameters
Uses the same configuration table as CML	Uses the same configuration table as IML
No compiler required	External compiler application required
Cannot access any C functions	C, C++ functions can be accessed
Less efficient	More efficient and faster processing
Debug switches can be used	No debug switches
Can validate procedures before running program file	Cannot validate entire procedure, specifically embedded C or C++ code
The task interprets every line of the procedures as they are executed at run time. FactoryLink loads the IML program files into memory at startup. Each time the IML procedure is triggered, the task interprets the procedure instructions and then performs the actions required.	At startup, the CML program files are compiled into executable C source code files and linked with FactoryLink and the externally supplied libraries. When CML procedures are triggered, the CML executable files are invoked. A binary executable is created for each domain: Shared domain: /{FLAPP}/SHARED/CML/CSHARED.EXE User domain: /{FLAPP}/USER/CML/CUSER.EXE The original .PRG file remains unaltered.

CONFIGURING MATH AND LOGIC

The Math and Logic folder contains the tables and text editor functions that provide access to configure the Math and Logic programs. The folder contains the following configuration tables that you will complete in the listed order:

- Math and Logic Variables table
- Math and Logic Triggers table
- Math and Logic Procedure table
- Math and Logic System Makefile table (CML only)
- Math and Logic Domain Makefile table (CML only)

IML is preconfigured in FactoryLink. If you are using CML, you must add the CML task to the System Configuration Table. Removing the IML task is not required.

Math and Logic Variables

Math and Logic variables are tags that can be global to FactoryLink or local to a particular procedure or program. A tag must be defined before it can be used in a procedure. All Math and Logic variables (tags) can be defined during the editing process while creating procedures or in the Math and Logic Variables Information table. This table must contain every tag referenced in the .PRG files. Failure to list a tag in this table results in validation errors and run-time errors.

- **Global variables** in Math and Logic are known as tags in other FactoryLink tasks. Any tags defined in this table are globally available to any FactoryLink process. To use any tag of an array, list only the zero tag (array [0]). The subscript must be zero (array [0]) to ensure proper functioning of Math and Logic.
- **Local variables**, specific to a file, can be defined within the program and cannot be used or referenced outside of the program.

For a complete explanation and attributes of each method, see the *Configuration Explorer Help*.

Field Description

Tag Name — Tag to be used in a Math and Logic procedure. Do not use reserved keywords or reserved tags as a tag name. If the tag is an array, specify 0 for each array dimension when entering its name; for example, batch[0][0].

Valid Entry: tag name

Valid Data Type: digital, analog, longana, message, float

Math and Logic Triggers

The Math and Logic Triggers table lets you define the trigger tags used to invoke (trigger) the Math and Logic procedures to run. An alternative to using a trigger to invoke a procedure is to call a procedure from another procedure using a function call. However, procedures that do not require triggers must still be defined in the Trigger Information table with no entry in the Trigger field.

Accessing

In your server application, open **Math and Logic Triggers > Math and Logic Triggers Information**.

Field Descriptions

Trigger Tag — Tag whose value can trigger a Math and Logic procedure.

- Valid Entry: tag name
- Valid Data Type: digital, analog, longana, message

Procedure — Unique name of the Math and Logic procedure exactly as you will enter it in the procedures (proc) statement within the program file.

- Valid Entry: alphanumeric string: 1 to 16 characters, case-sensitive, cannot be the same as a defined tag name and must begin with an alphabetic character.

Mode — Determines how the Math and Logic procedure instructions are executed.

- Valid Entry: INTERPRETED or COMPILED (in all uppercase, all lowercase, or initial caps)

Description — Indicates the intended use of the procedure.

- Valid Entry: alphanumeric string; 1 to 80 characters

Note: You can also define a trigger when you add a procedure.

Math and Logic Procedures

The Math and Logic Procedure table is an editor that lets you write the programs and procedures required by the application. A program file is a text file containing the text of one or more Math and Logic procedures.

Note: It is strongly recommended that you test all new programs and procedures in a test environment before launching them in an actual plant environment.

The procedures within a program file can be totally unrelated in functionality as they are individually invoked by the predefined trigger or a function call embedded in another procedure. All procedures in a program must be defined as either an IML or CML procedure.

At run time, procedures can be either:

- Triggered in response to changes in the real-time database
- Executed directly from within another Math and Logic procedure

A program file must contain at least one procedure, called the main procedure, that fits two criteria:

- The main procedure has the same name as the name of the program file, minus the extension. For example, the program file MYPROC.PRG must contain a main procedure with the name MYPROC, myproc, MyProc, or some other variation. Program file names are not case-sensitive.
- The procedure name entered in the Math and Logic Triggers table must be exactly the same as the name entered in the program file, including upper and lowercase characters.

Accessing

In your server application, open **Math and Logic > Math and Logic Procedures**.

Creating a New Program File

1. To create a new program file, expand the **Math and Logic Procedures** folder, right-click **Math and Logic Procedure - Shared** and select **New Prg file**. The New Math and Logic Program File dialog box appears.
2. Type a name for the program file name, and select either **Interpreted** or **Compiled** mode. Enter the name of the tag to trigger the program.
3. A new program, when saved, is automatically stored at: c:\{FLAPP}\{FL DOMAIN}\procs. It is essential that the file remains in this directory.
4. Expand the **Math and Logic Procedure - Shared** folder and then expand the program name. (A new procedure without the extension appears under the program name.) Open the program. The program file displays with the procedure definition statements (PROC, BEGIN, and END) inserted into the program file.

Adding New Procedures

1. Open the program file, position the cursor at the beginning of the line where you want to add the new procedure, and then click **Insert Procedure**  .
2. In the Insert Procedure dialog box, type the Procedure Name, the Trigger tag name (if applicable), select the mode (either **Interpreted** or **Compiled)**, and click **OK**. A template is inserted in the file to assist you with writing the procedure.
3. Add code to the template information to create the new procedure.

PROGRAM FILES AND PROCEDURES

Define procedures using the following general form (with no arguments):

```
PROC name
BEGIN
 .
 .
 .
END
```

Procedure definition statement

Body of procedure, declarations, and statements

Each procedure definition statement or **proc** statement starts with the word PROC, followed by the unique name of the procedure, followed by any arguments (parameters) the procedure requires. Any procedure, except the main procedure for the file, can have arguments. Place the keyword BEGIN on the next line.

For example:

```
PROC name (type name1 [,type name2])
BEGIN
 .
 .
END
```

where

type	Is SHORT, LONG, FLOAT, or STRING.
name1	Is the name of a variable, constant, or tag name
name2	Is the name of a variable, constant, or tag name other than name1

Table 14-2 shows the components used in writing procedures.

Table 14-2 Procedure Components

Component	Description
PROC name	**PROC** is a required statement that identifies the unique, case-sensitive name for the procedure. Procedure names are case-sensitive. The program file names must be in lowercase and have the .PRG extension (filename.prg). The .PRG extension is added automatically to the file name when defined. Program file names are not referenced in any FactoryLink tables, but all procedure files must be listed in the Math and Logic Triggers Information table. While procedure names are case-sensitive, the names of the program files the procedures are contained in are not case-sensitive. See “Coding Guidelines” on page 307 for additional information on writing programs and procedures.
BEGIN	**BEGIN** is a required statement that starts the body of the procedure.
END	**END** is a required statement that concludes the body of the procedure.
Comments	Comments are text to annotate a program and are ignored during execution. Comments begin with the pound sign (#) and end with the end-of-line. They can be nested or written on lines by themselves. If the line of code to be annotated is short, they can be written on the same line as the code. Always include comments in procedures for later reference. They provide programmers with information about the intended function of the procedure. Comments are also useful on variable declaration lines to explain how the variable is to be used and its value limits. Standard coding practice calls for one comment for approximately every five lines as well as at the top of any loop and before a function call.
Constants	A constant is a numeric or character value that remains unchanged during the execution of a program. Constants can be used in a calculation anywhere a numeral can be used and are faster to use in calculations than variables. See “Constants” on page 310 for additional information on using constants.
Declarations	Declarations create storage in memory for values used in a procedure. See “Declarations” on page 314 for additional information on using declarations.
Expressions	Expressions are sets of arguments (operands) that resolve to exactly one value. See “Expressions” on page 321 for additional information on using expressions.

Table 14-2 Procedure Components (continued)

Component	Description
Operators	Operators are symbols and/or keywords used in expressions to specify the type of operation to perform, such as adding or dividing. See “Operators” on page 321 for additional information on using operators.
Statements	Statements are instructions that describe mathematical and/or logical operations to perform in a specified order. See “Statements” on page 327 for additional information on writing statements.
Directives	Directives are symbols used in statements that determine the type of statement to perform. See “Directives” on page 330 for additional information on using directives.
Tags	FactoryLink tags used in the procedures are available to other tasks.
Local Variables	Local variables are variables that are used only within the procedure.

Figure 14-1 Sample Procedure

```
# declare global variables
DECLARE SHORT init_flag        # decision flag
PROC EXAMPLE
BEGIN
   DECLARE SHORT _n            # declare local variables
   DECLARE SHORT tub_array[100]
   # Do initialization procedure, but only once
   init_flag = 0               # initially, init_flag = 0
   IF NOT init_flag THEN
      # Initialize database variables
      pressure == 10           # force-write pressure in
                                 product tubs
      tubs = 20                # number of tubs to fill
      tempset = 71             # degrees Celsius; “warm”
      # Set tub_array[_n] = _n for _n =0,1,...,(tubs-1)
      _n = 0                   # start loop index at 0
      WHILE _n < tubs
         tub_array[_n] = _n
         _n = _n + 1
      WEND
   init_flag = 1               # done, so set init_flag ON
   ENDIF
END                            # PROC EXAMPLE
```

Coding Guidelines

- Always start the procedure with a BEGIN statement and conclude it with an END statement.
- The maximum line length is 1023 characters. Running a procedure with lines longer than 1023 characters can cause unpredictable validation results; the procedure may validate even though no errors occur. Math and Logic will not function properly while running such a procedure.
- For each IF statement, enter a matching ENDIF, properly nested.
- Show all keywords, such as IF, THEN, ELSE, and ENDIF, in uppercase characters to distinguish them from tag names. Keywords are not case-sensitive, but tag names are.
- A local variable (tag) can be declared in the program. If it is added to the top of the file above the initial BEGIN statement, it is available to all procedures. If the local variables are added at the procedure level, then they are only available to the procedure in which they were declared. To differentiate local variables from tag variables, begin the local variable names with "_".
- Global variables (tags) are added to a procedure by typing the tag name in the procedure. Highlight and right-click the tag name. Select **Add to Tag List**. The FactoryLink Tag Editor dialog box appears providing definition of the tag. For more information, see the *Configuration Explorer Help*. The tag color changes to blue when the definition is completed.
- Once defined, the tag name appears in the Xref Table, the Tag Browser, and the Object Table in addition to the Math and Logic Variables table. Global variables (tags) can be added at any time to the Math and Logic Variables Information table and then typed into the procedure. Type the variable (tag) name, and the variable text color changes from black to blue indicating it is already defined.
- Math and Logic can operate in either the Shared or the User domain. Use the Shared domain when all tasks or users must share the same Math and Logic data. If a Shared tag is used in both Shared and User procedures, it must be referenced in both the Shared and User Math and Logic Variables Information tables. By default Configuration Explorer displays only the Shared domain. To view both domains, right-click the application name and select **Shared+User**.
- User-inserted markers and error markers have the same appearance. Markers can be toggled with the shortcut <Ctrl + F2> on a specific line, or added to many lines using the find function. All previously set markers are erased when the validate function is performed. Save the procedure after you finish making changes.
- To avoid confusion and a possible error, do not give any two procedures, tag names, variables, or constants the same name, even if the case is different. Local variable names translate directly into C code when compiled. Even if you are using IML, it is important to understand this so that you develop procedures that can be compiled later if needed. If a

local variable is the same as a variable or function in another module or library, conflicts will occur at compile time. For CML procedures, the unique naming is limited by the effects of the compiled C code. For example, special characters ($.*) become "_" by the parsing routine.

For example, all of the following declaration statements become **declare short lu_lu** with potentially confusing results, such as duplicate definition errors, or the changes in one variable get reflected in another:

```
declare short lu$lu
declare short lu@lu
declare short lu_lu
```

- The number of tags, triggers, and programs you can define is limited only by the amount of available memory, the operating system, and an optional compiler (compiled mode only). Using CML requires a compiler program in addition to the FactoryLink software. Using IML does not have compiler requirements.
- The Microsoft® Visual C++ .NET compiler is the compiler to use with the FactoryLink CML processing. See the "Supported Layered Products Information" section in the *Installation Guide*. Refer to the documentation supplied with the compiler for details on the compiler limitations for your system.
- Math and Logic does not provide return codes for developer-defined procedures; therefore, the task cannot set a variable's value to the return code from a procedure call.
- After you finish typing the procedure, you must validate it to check for syntax errors. Click **Validate** to verify the syntax, such as matching braces, parentheses, and brackets, and the correct use of operators. The correct definition of local and global variables (tags) is checked plus the essential keywords are present (BEGIN, END, PROC). If no errors exist, the system reports nothing. If errors exist, red triangle markers display in the left hand margin for each line with an error. Correct the errors and revalidate the program.
- Math and Logic reserves a set of keywords for use in procedures. Because these keywords have predetermined meanings, they cannot be used as procedure names, local or global variable names, constant names, or tag names. The keywords are not case-sensitive. Do not write procedures that use forms of reserved keywords as names because they may cause unpredictable system behavior during execution.
- Because of the way floating point values are rounded and stored, you should not compare floating point values for equality in Math and Logic (or any other programming language).

Math and Logic Reserved Keywords

abs	declare	dosetdir	get_tag_types	or	sqr
add_tag	***default***	dosetdrive	***goto***	pop	***static***
alltrim	dig	dosub	gt	popdbvar	status
ana	div	dosubstr	i2f	popflp	string
and	do	dosys	idl	popint	***struct***
argcnt	doabs	dotrans	in	popstr	substr
Argcount	doadd	dotrim	***if***	poplong	***switch***
Argvect	doand	***double***	include	pow	system
asc	doasc	dounlock	init	print	tan
asm	doband	doupper	input	proc	Task_desc
auto	dobneg	doxor	instr	process	Task_name
begin ({)**	dobor	***else***	***int***	procnam	Task_id
break	dochr	end (})**	l2f	push	then
call	dodiv	endif	l2s	pushdbvar	tos
case	doeq	endwhile	lana	pushflp	tosflp
cbegin	dogetarg	***entry***	le	pushint	tosint
cend	dogetdir dolte	equ	len	pushlong	toslong
cfunc	dogetdrive	exit	line	pushstr	tosstr
changed	dogt	exp	load_trig	***register***	trace
char	dogte	***extern***	lock	resetstack	trprint
check_security	doinstr	f2s	log	***return***	trim
chk_prot_bit	dolenstr	fatal	loge	rnd	***typedef***
chkz	dolock	fixtype	***long***	run time	***union***
chr	dolower	fl_cmlpp	lower	s2f	unlock
ckalloc	dolt	fl_tagname_to_id	lt	s2i	***unsigned***
clrstack	doltrim	***float***	ltrim	savestr	upper
cml_force_math	domod	flp	main	setdir	void
cmlpp_listconst	domul	***for***	msg	setdrive	wait
cmlpp_options	doneg	ge	mod	***short***	wend
continue	doneq	getarg	mul	shutdown	***while***
connect	donot	getchng	nalloc	short_pause	xor
cos	door	getdir	ne	***signed***	
ct_load	dopow	getdrive	nfree	sin	
debuge	doprint	get_entry	not	***sizeof***	

* The reserved keywords in ***boldface-italic*** type are C keywords reserved by the C compiler. Program files cannot use these C keywords. Other keywords may exist; refer to the user manual supplied with the C compiler in use.

** The keyword **begin** is interchangeable with the opening brace ({), and the keyword **end** is interchangeable with the closing brace (}) inside Math and Logic programs.

Constants

A constant is a numeric or character value that remains unchanged during the execution of a program. Constants can be used in a calculation anywhere a number can be used and are faster to use in calculations than variables.

Constants are especially useful in applications when the boundary value of a loop or array must be modified. When the constant is modified, its value only has to be changed in one place within the application rather than many different places.

For example, a factory upgrades from three drying beds to five and the constant BED_MAX is used as:

- A loop index—to index through the operations on the groups of beds
- An array index—for the array containing information on each bed
- As a limiting factor on the number of beds polled

The value of BED_MAX can be modified from 3 to 5, thus preventing the need for massive search-and-replace operations on hard-coded values.

The various types of constants are discussed in this section.

Numeric Constants

Numeric constants can be assigned to digital, analog, long analog, or floating-point tags as well as to numeric local variables. Constants can be used in expressions wherever a numeric operand or argument is valid, provided they are not the objects of an assignment operator.

Because constants cannot take on new values, they must never be placed on the left-hand side of an assignment operator.

- **Integer constants**—You can assign integer constants to tags and local variables.

  For a tag, its data type must be one of the FactoryLink data types digital, analog, or long analog, and its value must be an integer.

  For a local variable, its data type must be one of the local variable types short or long, and its value must be an integer.

  Integer constants can be represented in binary, decimal, octal, or hexadecimal notation:

Binary	Strings of 0s and 1s in which the first two characters are either 0b or 0B (to indicate base-two representation).
Decimal	Strings of any digits 0 through 9 with the first digit either nonzero, 0d or 0D (to indicate base-10 representation).
Octal	Strings of any digits 0 through 7 with the first digit a 0.

Hexadecimal — Strings containing any combinations of the digits 0 through 9 and/or the characters A through F or a through f, in which the first characters are 0x or 0X (to indicate base-16 representation).

For example, to define the local variable **_length** as 28, use any of the following definitions:

Notation	Definition
Binary	_length = 0b11100
Decimal	_length = 28
Octal	_length = 034
Hexadecimal	_length = 0x1C

Furthermore, some values are too large to be represented as short ANALOG values and must be represented as LONGANA values. Any integer constant to be represented as a LONGANA (long integer) data type must be following by a trailing L.

The following value ranges must be represented as LONGANA values:

Notation	Value Range
Decimal	x < -65536 and x > 65535
Octal	x> 177777
Hexadecimal	x> 0xFFFF

For example, if a constant is to be larger than 65535, place a trailing **L** after the number to indicate long analog representation, as follows:

Notation	Long Analog Representation
Decimal	_upperlim = 123456L
Hexadecimal	_maxvalue = 0xFFFFFFFFL

Minimum and maximum long analog values can range between -2,147,483,647 and 2,147,483,647.

- **Floating-point constants**— Use standard floating-point notation or exponential notation to represent floating-point constants. Floating-point constants are strings of any digits, 0 through 9, that either contain or end in a decimal point.
- **Exponential constants**—Exponential constants are strings of any digits, 0 through 9, with an E, E-, e, or e- preceding the exponential portion of the value.

The following table shows numerals represented by numeric constants in the various notations just described.

Binary	Ob101	Ob001	Ob111
Decimal	12908	562334L	10
Octal	0123	033	05670222L
Hexadecimal	Ox45AB	OxOaOd	OX7CEFOAF4L
Floating-Point	465.95	0.0	24567.90667
Exponential	9780e12	332e-4	54221E234

String Constants

A string is a sequence of ASCII characters enclosed in double quotation marks (“ ”). String constants can be from 0 to 79 characters and the ending character must always be the final character in the string. For example, the string “ABC” consists of the characters A, B, C, in that order. An empty string has no characters and is represented as a space enclosed in double quotation marks. If an operator enters more than 79 characters as the value of a message, the task truncates the string to include only the first 79 characters.

You can assign string constants to message-type tags or string-type local variables. Math and Logic supports operator input in both IML and CML.

In string constants, the single backslash (\) character introduces print-formatting characters. The Math and Logic parser recognizes the single backslash as a signal that a print-format character (an escape code) follows. Therefore, the string “\” causes a parsing error during Math and Logic processing because nothing follows the backslash. If a backslash is required within the string itself, use a double backslash (\\). The following table lists the meanings of the print-formatting characters in Math and Logic.

Character sequence	Meaning (print result)
\b	backspace
\f	form feed
\t	horizontal tab
\v	vertical tab
\\	backslash
\”	double quote
\’	single quote
\r	carriage return
\n	new line (carriage return/line feed combination; end-of-line character)

Other special ASCII characters, such as nonprinting control characters (for example, the escape character), are sometimes needed as constants. Use the **chr** function to refer to these characters.

To store ASCII data, including nonprinting ASCII characters, as string constants, enter the ASCII code in a call to the built-in Math and Logic function chr, which has the following format:

```
chr(xx)
```

where `xx` is the ASCII code to generate the character.

For example:

```
x = chr(27)   # sets the string variable x to the escape
                character
x = chr(124)  # assigns to x the "vertical bar" symbol (|)
```

Refer to any table of standard ASCII character codes to determine the proper ASCII value of any character. The following examples illustrate the use of string constants.

Constant	Resulting String
CHR(27)	ESC character
"ABC\n"	'A', 'B', 'C', new line, NUL
"MENU"	'M', 'E', 'N', 'U', NUL
"\""	Double quotation mark, NUL

Use double quotes to include special characters inside quoted strings.

Refer to the system software documentation supplied with the operating system for specialized information about ASCII characters and the details of string handling, such as values of the machine's character set.

Symbolic Constants

A symbolic constant is a name you define to represent a single, known numeric value. You can define a symbolic constant using either of two formats—with an equal sign or with a space to separate the name and value. In the example, a symbolic constant PI represents the value 3.14159; thereafter, the constant PI can be used wherever needed in place of the value 3.14159.

Format	Example
`CONST` *`name value`*	`CONST PI 3.14159`
`CONST` *`name=value`*	`CONST PI=3.14159`

Declarations

Declarations tell a procedure:

- A variable or a constant is to be created or a variable or a procedure is to be referenced.
- The scope of the created variable or constant or the referenced variable or procedure. Scope is that part of a program in which a variable, constant, or procedure can be used. This varies according to where the declarations take place. Math and Logic uses two categories of scope:
  - **Block (Local) scope**—Starts at the declaration point and ends at the end of the block containing the declaration.
  - **File (Global) scope**—Starts at the declaration point and ends at the end of the source file.

Procedure Declarations

A procedure declaration identifies a procedure either defined later in the current program file or is referenced (called) by a procedure in the current program file. Use one of the following forms to declare a procedure depending on whether or not the procedure will accept arguments:

```
DECLARE PROC name
```

or

```
DECLARE PROC name (type[.type])
```

If a procedure is to take arguments, use the second form given above. Only the data type of each argument is given in a procedure declaration. The data type of each argument is the same as the original local variable (SHORT, LONG, FLOAT, STRING).

The number of arguments in the declaration, the order the arguments are entered in, and their data types must match the procedure definition. The procedure declarations are convenient when a custom-written procedure must refer to another custom-written procedure that has not yet been encountered because it is contained within another program file or occurs later in the same program file. Procedure declarations are not required when the procedure called is displayed in the same file but before the current procedure.

The following example shows how procedure declarations affect procedure calls:

```
PROC A
BEGIN
 .
CALL B                  Because Procedure B has not been declared and does not
 .                      appear before Procedure A, this call is not allowed.
 .                      Procedure B must be declared first.
END
PROC B
BEGIN
  .                     Because Procedure A is displayed before Procedure B, this
CALL A                  call is allowed.
 .
END
```

Using the same example, by declaring PROC B above the definition of PROC A, then PROC B can be called:

```
DECLAREPROCBPROCA    Declare PROC B.
BEGIN
 .
CALL B
 .
END                  PROC B can be called here because  it was declared
PROC B               previously.
BEGIN
 .
CALL A
 .
END
```

You can also call a procedure or function defined in another program file. If no triggered procedures exist in the referenced program file, then the Math and Logic Triggers table must contain an entry for that file.

In the following example **Func1** is called in **PROC1.PRG**, but it is defined in **PROC2.PRG**. Therefore, PROC2 requires an entry in the Math and Logic Triggers table.

```
PROC1.PRG

DECLARE PROC func1
PROC PROC1
BEGIN
 .
 .
CALL func1
 .
 .
 .
END
```

```
PROC2.PRG

PROC PROC2
BEGIN
 .
 .
END

PROC func1
BEGIN
 .
END
```

Constant Declarations

Constants are shared by all procedures and must be declared before any procedure in which they are used; therefore, place constant declarations above the procedure statement of the first procedure within the program file the constant is referenced in. Only one constant can be declared on each line.

Variable Declarations

Variables can be declared in a Math and Logic program as procedure variables or as tags. Variables declared as procedure variables are used to store values used only by Math and Logic to perform operations. These values cannot be used by other FactoryLink tasks because they are not tags in the real-time database.

Although procedure variables are not tags in the real-time database, they are still represented in system memory and can be saved and opened repeatedly or printed during the running of those procedures that can open them.

Use the following guidelines to determine whether to declare a variable as a procedure variable or as a tag:

- If the variable is opened from an external source, declare it as a tag.
- If the variable is a trigger for any procedure, it must be declared as a tag defined as a trigger tag with an associated trigger tag name.
- If the variable is used only by Math and Logic and must be accessible by all of the procedures within a program file, declare the variable as a procedure variable with a global scope by declaring it outside the first procedure in the program file.

- If the variable is used only by Math and Logic and is used only within a particular procedure, declare the variable as a procedure variable with a local scope by declaring it inside that procedure.

Local procedure variables can be of one of the following data types:

- SHORT (digital)
- SHORT (analog)
- LONG (long analog)
- FLOAT (float)
- STRING (message)

Variables declared inside a procedure must have different names from variables declared outside of a procedure. The case of a variable name is significant.

- A variable name cannot begin with a digit (0-9).
- Variables cannot be initialized at declaration.
- Arrays cannot be passed as arguments to a procedure, but individual array tags can.

Local Procedure Variables—Declare local procedure variables immediately after the BEGIN statement. A local variable declaration must precede all other instructions in a procedure. Local variables are declared one data type to a line in statements similar to:

```
DECLARE short _itotal
```

Initialized Value—Each time a procedure is called in the interpreted mode, a new instance of each local variable is created and the value of each variable is initialized to 0. Each time the executable is run in the compiled mode, the value of each local variable is initialized to 0, which redefines the variable. When a procedure is completed, variables defined inside the procedure are destroyed.

When large numbers of local variables are declared in a program file and are meant to be accessible to all the procedures in that file, performance can be improved by placing the declarations at the top of the file in which the procedures are stored before the start of the first procedure. This makes the declarations global to the program file.

A local procedure variable may be declared as a scalar local variable or as a local array.

Scalar Local Variable—If declared as a scalar local variable, the declaration has the following form:

```
DECLARE type name
```

where

type Is one of the following:

SHORT	signed short integer
LONG	signed long integer
FLOAT	double-precision floating-point number
STRING	ASCI character string of 1023 characters in CML and 1024 in IML.

name Contains only alphabetic characters (A-Z, a-z), digits (0-9), periods (.), dollar signs ($), at-signs (@), and underscores (_).

Has a maximum length of 30 characters.

Does not have a period as its first or last character.

Does not have a digit as its first character in a name.

In the compiled mode, the periods (.), dollar signs ($), and at-signs (@) are all converted to underscores (_) in the resulting C source file.

For example, the variable names,

```
.temp
$temp
@temp
```

all equal _temp when they are translated into C source code; therefore, avoid variable names with periods, dollar signs, and at-signs, in case you need to convert to CML in the future.

Separate the names with commas, as shown in the following example, to declare more than one variable of the same type on the same line:

```
DECLARE SHORT _s1,_s2,_s3   # loop & array indices/3D array
```

However, we recommend variables be declared one to a line with comments on the same line after each declaration briefly describing the use of the variable.

Consider the following examples:

```
DECLARE SHORT_itotal       # total number of vats heated
DECLARE FLOAT_ftempm       # mean temperature held in vats
```

Local Array—A local variable can also be declared as a local array. A local array represents a set of values of the same type. An array is declared by specifying the size or dimension of the array after the array type. An array can have a maximum of 16 dimensions. An array with more than one dimension may be thought of as an array whose tags are also arrays rather than scalar variables; each additional dimension gains the array another set of row and column indices. Each of the dimensions, which must be constant, are enclosed in brackets.

Use one of the following forms to declare a local variable as a local array:

```
DECLARE SHORT _week[7]          # days of the week
```

or

```
DECLARE SHORT _cal[12][31][10] # ten-year calendar array
```

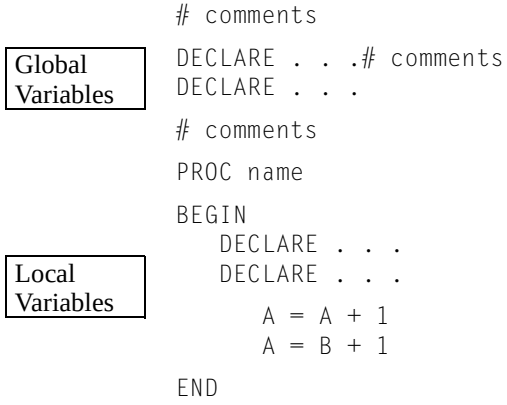
The second form defines a three-dimensional array. The total number of tags in array `_cal` is the product of the size of each dimension.

Local variable arrays function like scalar local variables in many ways except neither an array nor an array tag can be passed as an argument to a procedure.

Global Procedure Variables—You must declare global variables outside of any procedure that references them. For Interpreted Math and Logic, declare global procedure variables before the first procedure definition in a program file. For purposes of validation, declare global variables in each program file they are used in. After the first invocation, they retain their values across procedure calls.

Generally, use a variable, constant, or procedure after its declaration point in a program; therefore, where variables, constants, and procedures are declared in a Math and Logic program depend on their intended scope.

The following model shows where you declare global and local variables.



Limitations

The 64K barrier under segmented architectures, such as Microsoft Windows, presents a limitation on the size of some variable data in Math and Logic. Neither global nor local variable arrays or data items, such as string arrays or message/buffer data, both of which tend to become large, may exceed 64K. Items declared larger than 64K will, nevertheless, be allocated only 64K under Microsoft Windows; no compile-time or table-entry checking is planned to limit the size of declarations because of the multi-platform nature of the current FactoryLink software system. Note also that the index (sizing) value for a variable array is limited to 32K (32767); array dimensions must be declared so as not to exceed this limit.

Note these limitations when designing your application. Any global or local variables that must be larger than 64K should be partitioned logically during design so no data item as declared exceeds 64K. Declare several linked data items, if large buffers are needed in an application.

EXPRESSIONS

An expression is a set of operands that resolve to exactly one value. An expression consists of some combination of the following tags:

- Operators (symbols or keywords that specify the operation to be performed)
- Variables (tag names and procedure variables)
- Constants (symbolic, numeric, and string constants)
- Functions (user defined and library)

In an expression, parentheses and brackets are balanced and all operators have the correct number and types of operands. The following examples illustrate well-formed expressions (assuming the data types of each operand are valid with the operators):

```
5
X + 3.5
temp < 0 OR temp > = 100
outrange AND (valve1 = 1 OR valve2 = 1)
100*sin(voltage1 - voltage2)
“This is a message to the operator!”
```

OPERATORS

Operators are symbols or keywords that are used in expressions to specify the type of operation to be performed. Operators can be either unary or binary. Unary operators operate on only one operand at a time while binary operators operate on two operands at a time.

Math and Logic employs the following operator groups, arranged in alphabetical order:

- Arithmetic
- Bitwise
- Change-Status
- Grouping
- Logical
- Relational

These operators must be used in a particular sequence to get the desired results from a calculation. For information about the order in which the operations are performed in an expression, refer to “Calling Procedures and Functions” on page 338.

Arithmetic Operators

Arithmetic operators perform arithmetic operations on their operands. The following table illustrates arithmetic operators.

Operator	Type	Usage	Operands	Operation Name
+	binary	x + y	numeric	addition
+	binary	x + y	string	concatenation
-	binary	x - y	numeric	subtraction
-	unary	-x	numeric	negation
*	binary	x*y	numeric	multiplication
/	binary	x/y	numeric	division
^	binary	x^y	numeric	exponentiation
MOD	binary	x MOD y	integer	modulo (or modulus)

Place spaces before and after the keyword MOD to avoid confusion with variable names when the program parses the formula.

All arithmetic operators except modulo operate on any type of numeric operands, including floating-point. MOD functions with only integers. In the case of tag names, this means any combination of analog or long analog data types.

Math and Logic defines modulo as follows:

```
x MOD y
```

This operation returns the remainder after dividing x by y. The following examples illustrate arithmetic operations.

Operation	Results
17/5 = 3	Returns quotient of 3; remainder is lost
17 MOD 5 = 2	Returns remainder of 2; quotient is ignored
17.0/5 = 3.4	Result is converted to floating-point; returns quotient and remainder

Bitwise Operators

Bitwise operators compare and manipulate the individual bits of their operands. Math and Logic resolves all operands to integers. The following table illustrates bitwise operators.

Operator	Type	Usage	Operands	Operation Name
!	unary	!x	integer	bitwise one's complement
&	binary	x & y	integer	bitwise AND
|	binary	x | y	integer	bitwise OR (inclusive OR)
~	binary	x~y	integer	bitwise XOR (exclusive OR)

Math and Logic defines bitwise operators as demonstrated in the following table.

Operation	Bit Value	Return Value
9 & 10	9 = 1001 10 = 1010 1000	8
9 | 10	9 = 1001 10 = 1010 1011	11
9 ~ 10	9 = 1001 10 = 1010 0011	3
9 ~ - 10	9 = 0000000000001001 -10 = 1111111111110110 1111111111111111	-1
! 9	9 = 0000000000001001 bitwise flip produces 1111111111110110 !9 = 1111111111110110	-10

Change-Status Operators

The change-status operator checks whether the value of a tag has changed since Math and Logic's last read operation of that tag. If the change-status bit for Math and Logic has been set for any reason, including a forced write, the operation returns a value of TRUE (1). It is important to understand that the use of the change-status operator itself resets the tag change-status bit with respect to Math and Logic. Consequently, do not perform change-status operations on a tag more than once in Math and Logic. If you need to use the result of a change-status operation in multiple places, assign the value to another tag and use that tag in your calculations.

The following table illustrates the characteristics of change-status operations.

Operator	Type	Usage	Operands	Definition
? changed	unary	? x changed x	tag name	The operation returns TRUE (1) if the value of x has changed since it was last read or change-read by Math and Logic and FALSE (O) if it has not.

Do not enclose the tag name (operand) in parentheses when checking change status. The construct ?(x) is misinterpreted by Math and Logic in this context and does not produce the desired result. Always use the construct ? x or (changed x).

The following table illustrates two change-status operations.

Operation	Results
y = y + ?x	Increments the value of y by 1 whenever the value of x changes.
If (changed my_tag) then call proc my_proc Endif	Initiates the procedure my_proc whenever the value of my_tag changes.

Do not perform change-status operations on tags being used as procedure triggers (trigger tags). This may prevent the corresponding procedure(s) from being triggered at the proper time. This is because checking the change status of the tag resets the change bit for that tag.

Grouping Operators

The following table illustrates the special grouping operators.

Operator	Name	Use
( )	Parenthesis	Use these to group sub-expressions. Their main purpose is to override the precedence of operations by forcing the evaluation of other operations first. Also, use parentheses to enclose arguments being passed to a function or procedure.
[ ]	Brackets	Use these to enclose array indices. Use multiple pairs of brackets for double- or triple-indexed arrays.
,	Commas	Use these to separate the arguments (if more than one) being passed to a function. Also, use commas between types in procedure declarations and between type-argument name pairs in procedure definition header statements (proc statements).

Logical Operators

Logical operators test operands for TRUE (nonzero) or FALSE (zero) values and return a result of 1 (TRUE) or 0 (FALSE). Math and Logic resolves all operands to numeric form. The following table illustrates logical operators.

Operator	Type	Usage	Operation Name	Operation Definition
NOT	unary	NOT x	logical NOT	If x is zero, result is 1. If x is nonzero, result is 0.
AND	binary	x AND y	logical AND	If x != 0 and y != 0, result is 1. If x or y (or both) are 0, result is 0.
OR	binary	x OR y	logical OR	If x or y (or both) are != 0, result is 1. If x is zero and y is zero, result is 0.

Place spaces before and after the keywords AND, NOT, and OR to avoid confusion with variable names when the program parses the formula. The following table shows logical operators.

Operation	Return Value	Operation	Return Value
NOT 3	0	0 AND 0	0
NOT 0	1	1 AND 2	1
NOT - 1	0	0 OR 2	1
0 AND 1	0	0 OR 0	0

Relational Operators

Relational operators compare one numeric operand with another and generate a result that describes the outcome of the comparison. The result of a given comparison is 1 (TRUE) or 0 (FALSE). Math and Logic resolves all results to numeric form. The following table illustrates relational operators used in comparisons.

Keyword	Type	Usage	Operation Name	Operation Definition
=	binary	x = y	equal to	If x = y, result is 1. If x != y, result is 0.
!= or <>	binary	x != y	not equal to	If x = y, result is 0. If x != y, result is 1.
<	binary	x < y	less than	If x < y, result is 1. If x >= y, result is 0.
>	binary	x >y	greater than	If x > y, result is 1. If x<= y, result is 0.
<=	binary	x <= y	less than or equal to	If x <= y, result is 1. If x > y, result is 0.
>=	binary	x >= y	greater than or equal to	If x >= y, result is 1. If x < y, result is 1.

Given the short analog variable x = 3, the results of various relational operations done using **x** as an operand are shown in the following table.

Operation	Return Value	Operation	Return Value
x < 10	1	x = 4	0
x < 3	0	x <= 3	1
x > 2	1	x <= 4	1
x > 3	0	x <= 2	0
x != 4	1	x >= 2	1
x != 3	0	x >= 3	1
x = 3	1	x >= 1	1

STATEMENTS

A statement is an instruction that describes mathematical and/or logical operations to be performed in a specified order. Statements can be one of three types: assignment, control, procedure call.

Assignment Statements

Assignment statements assign values to Math and Logic procedure variables or tags and can have either of the following forms, where = and == are the assignment operators. Whether in a formula or within a procedure, assignment statement are written with the variable to be changed on the left-hand side of the assignment operator and the term or expression whose value should be taken on the right-hand side. Math and Logic computes the expression **expr** and assigns the result to the procedure variable or tag.

A significant difference exists between the two assignment operators.

The following examples use the tags **fptemp** and **itemp** to demonstrate the difference:

x = expr	Only writes if value of **x** has changed. Valid for procedure variables and tags. Will not change the value of **x** unless it is different from the value in **expr**.
x == expr	Forced write, regardless of tag's present value. Turns on change-status flags for **x** regardless of whether its value actually changed or not. Valid only for tags.

Note: You can end an assignment statement with a semi-colon (**;**), if desired.

Control Statements

Control statements include instructions that determine when a block of code is to be executed. End a control statement line only with an end-of-line character, never with a semicolon.

There are two types of control statements:

- IF...ENDIF
- WHILE...WEND (ENDWHILE)

IF...ENDIF—An IF...ENDIF statement specifies an action is to be executed only if a specified condition is true.

Syntax—The IF...ENDIF control statement has the following syntax:

```
IF expr THEN   # Block to be executed if expr is true.
[ELSE]         # Optional block to be executed if expr
               # is false.
ENDIF
```

If the test expression of the statement is true, the THEN block is executed. If the test expression is not true and an optional ELSE clause exists, the ELSE block is executed.

The IF...THEN block is not optional and the THEN verb must immediately follow the test expression on the same line as IF. Each IF statement must be ended with an ENDIF statement on a line by itself. The following example illustrates the use of IF...ENDIF control statements:

```
# Example: (The semicolons in this example are optional.)
IF x = 1 THEN
   a = a + 1; # increment a by 1
   b = 20;
ELSE
   a = a - 1; # decrement a by 1
   b = 0;
ENDIF
```

WHILE...WEND—A WHILE...WEND statement specifies a block of code that is to be executed repeatedly until the test expression becomes false.

Syntax—The WHILE...WEND control statement has the following syntax:

```
WHILE test_expr     # Block to be executed while
                      test_expr remains true.
 .
 .
WEND
```

If the expression **test_expr** is false, the block is not executed. The block is executed while the expression **test_expr** is true.If the expression never becomes false, the loop never terminates until the operator or another run-time process forces the procedure to stop running. Ensure the value of **test_expr** can become false at some point in the loop’s execution to prevent the program from hanging.

The keyword ENDWHILE can be substituted for WEND. The following examples illustrate the use of WHILE...WEND control statements:

```
# Example 1:

n = 0
WHILE n < 10
   a[n] = -1
   n = n + 1
WEND


# Example 2:

fib[0] = x
fib[1] = y
n = 2
WHILE n < 100 AND fib[n-1] < 10000
   fib[n] = fib[n-2] + fib[n-1]
   n = n + 1
ENDWHILE
```

Indent conditionally executed blocks for readability; program execution is not affected.

Procedure Call Statements

Procedure calls are statements that cause specified procedures to execute. Procedure calls are used to reference custom-written procedures or library functions built into the FactoryLink system.

Syntax

See “Calling Procedures and Functions” on page 338 for more information on calling procedures.

Block Nestability

Blocks delimited with control statements can be nested, provided each IF or WHILE statement contains an appropriate matching ENDIF or WEND statement. Blocks cannot overlap and must be matched pairs entirely within any other blocks to which they are internal. Improperly nested logic causes unpredictable results. This example shows proper procedure nesting.

```
IF x = y THEN
   n = 0
   WHILE n < 10
      a[n] = 0
      n = n + 1
   WEND
ELSE
   IF x > y THEN
      a[x-y] = 1
   ELSE
      a[y-x] = -1
      IF alert THEN
         PRINT “FOUND IT \n”
      ENDIF
   ENDIF
ENDIF
```

Excessive nesting of blocks or procedure calls can cause the operating system to halt the procedure and return a Stack overflow error. If this occurs, either restructure the procedures to reduce the number of nesting levels or increase the stack size for Math and Logic.

Directives

Directives are symbols used in statements. Math and Logic recognizes the directives in the following table.

Terms used in this table are:

x	is a variable capable of being assigned a value. It may be tag name, a local variable, or a constant, but it may not be a function.
expr	stands for an arbitrary expression
block	refers to any sequence of consecutive statements

Directive	Usage	Meaning
=	x = expr	Assign expr to x (via database write or locally)
==	x == expr	Assign expr to x via a forced database write
if	if expr	Test portion of if statement
then	then block	(required) If test is TRUE, begin “then” block
else	else block	(optional) If test is not TRUE, begin “else” block
endif	endif	End of “if- then” or “if-then-else” block
while	while expr	While expr is TRUE, do “while” block
wend	wend	End of “while” block
begin (or “{“)	begin	Begin program
end (or *}*)	end	End program
call	call proc01	Execute a procedure and return (optional keyword)
proc	proc block	Define a procedure
system	x = system(* *)*	Execute a system call and return Enclose a system call appropriate to the operating system within the double quotation marks. Virtually any system call, such as the “dir” command, is valid.
declare	declare short x	Define a variable or declare a procedure
print	print “starting”	Print a line of text
lock	lock	Lock the database
unlock	unlock	Unlock the database

OPERATOR PRECEDENCE

Most high-level languages use relative operator precedence and associativity to determine the order procedures perform operations in. If one operator has higher precedence than another, the procedure executes it before the other. If two operators have the same precedence, the procedure evaluates them according to their associativity, which is either left to right or right to left and is always the same for such operators.

Because parentheses are operators with very high precedence, they can be used to alter the evaluation order of other operators in an expression.

The Math and Logic operators are divided into 10 categories in the following table of operator precedence. The operators within each category have equal precedence.

Unary operators associate from right to left; all other operators associate from left to right.

Precedence (1 is highest)	Category	Operator	Description
1	Change-status	?	Checks change-status of tags
2	Grouping	( )	Grouping or function call
		[ ]	Array subscript
3	Unary	-	Unary minus
		NOT	Bitwise !'s (one's) complement
		!	Logical negation
4	Binary	&	Bitwise AND
		|	Bitwise OR
		~	Bitwise XOR
5	Multiplicative	^	Exponentiation
		*	Multiplication
		/	Division
		MOD	Modulus (remainder)
6	Additive	+	Binary plus (addition)
		-	Binary minus (subtraction)
7	Relational	<	Less than
		<=	Less than or equal to
		>	Greater than
		>=	Greater than or equal to
8	Equality	=	Equal to
		!=	Not equal to
9	Logical	AND	Logical AND
		OR	Logical OR
10	Assignment	=	Variable or tag assignment
		==	Forced write (tag only)

The following table illustrates operator precedence.

Example Statement	Explanation of Operator Precedence
x <5*y + 10 AND NOT zflag	According to precedence rules, the order of evaluation is NOT, then *, then +, then <, then AND. The program evaluates the expression as if it contained the following parentheses: (x < ((5*y) + 10)) AND (NOT zflag) To perform the addition before the multiplication, add parentheses to alter precedence, as follows: x < 5*(Y + 10) AND NOT zflag.
x * y/z	Because * and / have the same precedence and they associate left to right, the program performs * before /. The program evaluates the expression as if it contained the following parentheses: (x * y)/z.
x = NOT NOT y	The program evaluates the expression as if it contained the following parentheses: x = (NOT (NOT y)). This expression assigns the value 1 to x if y is nonzero. Otherwise, the expression assigns 0 to x and is equivalent to x = y ! = 0.
x = NOT ?y	The program evaluates the expression as if it contained the following parentheses: x = (NOT (?Y)).
x = NOT y = z	The program evaluates the expression as if it contained the following parentheses: x = ((NOT y) = z). This expression is different from x = NOT (y=z). The latter is equivalent to x = y ! = z.
a = 1 AND b ! = 2 OR c = 3 AND d ! = 4	The program evaluates the expression as if it contained the following parentheses: ((a =1) AND (b ! = 2)) OR ((c = 3) AND (d !=4)).
x = a = b < c	The program evaluates the expression as if it contained the following parentheses: x = (a = (b < c)). This expression assigns x the value of 1 if a has the same truth value (1 or 0) as the comparison b < c. Otherwise, the expression assigns a value of 0 to x.

Example Statement	Explanation of Operator Precedence
x & 1 | y & 2 | z & 4	The program evaluates the expression as if it contained the following parentheses: ((x & 1) | (y & 2) | (Z & 4). If this case, it does not really matter which | the program executes first, the program evaluates the expression from left to right.
f(2*x + 3, 4*y + 5) - g(x, y) /2	The program evaluates the expression as if it contained the following parentheses: f (((2*x) + 3), ((4*y) + 5)) - (g(x,y)/2).

DATA TYPE CONVERSION

Automatic conversion of any numeric data type (digital, analog, long analog, or floating-point) takes place when a variable of one type is assigned the value of another variable of a different data type. Use the simplest type of assignment statement necessary for the conversion to obtain the most efficient performance.

Include the new variable in a configuration table or in a program file depending on whether the original variable is a tag or is local to program operations. This will greatly simplify the debugging process should a problem occur during startup.

Create a new variable of a particular data type for accuracy in computation, such as floating-point, and initialize the new item to the current value of another variable of a different data type, such as a long analog. This conversion prevents a possible loss of accuracy in upcoming calculations. Use the new variable to do operations with other variables of the same type as the new variable.

Data type conversions are not often needed, but they can be useful in particular situations. Convert variables whenever the result requires the accuracy of the most precise data type involved or when incompatible operations are taking place between digital and analog values. Data type conversion can ensure the accuracy of the results of certain calculations with a few exceptions. The following guidelines indicate when and why data types should be converted.

Guidelines for Converting Data Types

Floating-point truncation If a floating-point value is assigned to a digital, analog, or long analog variable, Math and Logic truncates the fractional part of the floating-point value instead of rounding it up or down before assigning the value to the variable. This results in a loss of accuracy. This may be acceptable when the fractional values are not significant in the result of the calculation.

Non-zero conversion

Assigning an analog, long analog, or floating-point value to a digital variable converts any nonzero value to 1 and returns 0 if the value is 0. This can be useful when all one needs to know is whether a particular variable's contents are nonzero (for use as a trigger tag, perhaps), but it can result in a loss of precision in other situations. Because a digital data type holds only a single bit of data, the operation has only two possible results: if the assigned value is 0 after any fractional part is truncated, the digital variable takes on a value of 0; if the assigned value is nonzero after any fractional part is truncated, the digital variable takes on a value of 1.

Floating-point conversion

Assigning a floating-point value to an analog or long analog variable can result in a loss of precision [loss of least-significant bit(s)] when the system truncates the fractional portion of the value. The integer portion of the floating-point value might also exceed the maximum number of bits allocated to an integer which causes an overflow in the analog tag.

The maximum number of bits used by an integer depends on the data type of the tag or variable. Data types and maximum number of bits are as show in the following table.

Data Types	Maximum Number of Bits
SHORT (analog)	16
LONG (long analog)	64
FLOAT (floating-point)	64

Data type precision

When numeric data types are used in arithmetic operations (+, -, *, /), the result has the precision of the most accurate data type. If one of the data types is floating-point, the result is floating-point; otherwise, the result is analog. Digital and analog data types are internally represented as signed integers.

Overflow

Execution of arithmetic operations can result in an out-of-range value being placed into an analog or float variable. This results in a condition known as overflow [loss of most significant bit(s)] in that variable.

To avoid causing overflow, do not use calculations in your application that divide very small numbers by very large numbers, those that divide very large numbers by very small numbers, or those that divide a number by zero.

Before performing computations, ensure the results will be within the stated maximum and minimum ranges of the system itself; however, if you need to use larger analog values than the system can handle, use floating-points as a workaround; situations requiring numbers larger than the float representations possible on most systems will almost never arise.

It is recommended that you test for integer overflow (analog values) conditions and for floating-point overflow (float values) conditions.

Short to long analog conversion

Short analog values are simple to convert to long analog values.

Long Analog Value Overflow Correction Example—If temp is a 16-bit analog tag that currently has the value 30000, the erroneous result of the computation **2 X temp** will be -5536, instead of the expected value of 60000 because the result of this computation does not fit into the integer field allocated to the analog tag. It is not automatically converted to floating-point, so if you anticipate working with large values, it is wise to provide for the conversion of the value. Force this conversion by creating a floating-point variable **fptemp** and setting **fptemp = 2.0 X temp**.

Floating-point Value Overflow Correction Example—Although it is rare, overflow conditions can also occur in operations on floating-point data types. The result of an overflow is unpredictable and usually fatal. For example, division by zero always results in an overflow. If this happens, modify the programs involved so any variable about to be used as a divisor is tested for zero and provide for a method to skip the operation or to write an alert message whenever these conditions arise. If an overflow occurs, check string concatenation (joining) operations to determine if they are resulting in the assignment of inappropriate strings to floating-point values. This is discussed in String conversion and in Example 4 on page 336.

Array Conversion

If converting a local variable declared as an array, convert each array tag separately. If the conversion computation is within a loop, use one temporary variable of the new data type to represent the array index or create an array of the same size and of the new data type and load it with the converted values.

Examples of Data Type Conversion—The following four examples illustrate data type conversion:

Example 1—Converting analog values to digital

Let `digital_1` be a digital tag. The statement `digital_1 = 25.2` results in `digital_1` having a value of 1 (ON) because the assigned value after the fractional part is truncated is nonzero.

Example 2—Float truncation into integer

Let `analog_1` be an analog tag (16-bit integer). The statement `analog_1 = 47.1` results in `analog_1` having the value 47, not 47.1. Since `analog_1` is an integer, the decimal portion of the floating-point constant is truncated before the assignment.

Example 3—Appending numerics to messages

Let `mytag` be an analog tag with the value 99. `string1` is a message tag. The statement `string1 = "RPT" + mytag` results in `string1` having the value `RPT99`. `mytag` is converted to a string and then concatenated to the string constant `RPT`. The result `(RPT99)` is then assigned to `string1`.

Note: CML does not support appending numerics to messages.

Example 4—Equating a floating-point tag to the contents of message data.

Let `message1` be a message tag set to 1e308 (representing the number 10 raised to the power 308, a large floating-point constant stated in string form). Assume you set `message2`, defined the same way as `message1`, to a value of -1e-308 (representing the very small floating-point constant 10 raised to the power -308).

Let `float1` be a floating-point tag that receives the total of these two message tags. The statement `float1 = message1 + message2` which should add the two values, instead results in `float1` receiving an undefined value represented in the system as 1.#INF (is not float) or something similar, leading to an unpredictable result. This happens because the system performs string concatenation (the + operator acts as a concatenation operator in regard to string operands), which yields 1e3081e-308. The system stops converting at the second occurrence of **e** (discarding the -308 portion) and attempts to place into the variable **float1** the out-of-range value (10 raised to the power of 3081), which is too large to fit into the floating-point constant and is not the desired value.

Convert each of these values before adding them to prevent this type of error and avoid unpredictable system behavior. Create two conversion variables, **float1** and **float2**, and replace the statement above with the following statements:

```
float1 = message1        # get first user input
float2 = message2        # get second input
float1 = float1 + float2    # sum the inputs
```

The computations can then be carried on using only `float1`.

String conversion

A string type can be used in an arithmetic or logical operation if and only if the string is a sequence of digits including the natural logarithm constant **e**. The string is converted to a number before the operation is performed; conversion stops at the first nonnumeric character (except e). A string can be from 0 to 79 characters. If an operator enters more than 79 characters as the value of a message, the task truncates the string to include only the first 79 characters.

String-to-numeric conversion stops at the first nonnumeric character. In general, when alphabetic characters are used inside a string that contains only numeric data, the task sometimes interprets the characters differently from the way you intended. We recommend you set up some type of input checking if strings are to be routinely accepted as operands for computations.

A special case is when the letter **e** is included inside the string. During string conversion, **e** is interpreted as exponential, an indication the string of digits is to be considered written in scientific notation. See “Numeric Constants” on page 310 for more information about numeric representation. This allows the representation of larger numbers in a more compact format. It can, however, cause problems if you have not planned for limits on the type, number, and location of alphabetic characters inside these strings, and if you do not invoke logic to test the contents of the string for appropriateness before calling the conversion routine. See Example 4 on page 336 for an example and a description of the resulting workaround.

If the **+** (plus) operator is used in an expression with an operand of the string type, the other operand in the expression is automatically converted to a string (if it is not already a string), and the **+** operator concatenates the two strings. This type of operation is considered to be a string operation, not a logical or arithmetic operation. (See Example 3 on page 336.)

Calling Procedures and Functions

Two types of routines can be called from within an expression: developer-defined local procedures and library functions. Both procedures and functions may have values passed to them to use in their calculations.

Arguments

Arguments are values passed to a procedure for it to use in its computations. Arguments are input-only parameters.

Declare arguments by placing their types and names in the procedure definition statement, as shown in the example above. Local and global variable names and tag names can be used. The data type of the argument is the same as that of the original variable or tag (SHORT, LONG, FLOAT, STRING).

Math and Logic copies the values used as arguments so the procedure modifies the copies, not the original values of the variables or tags. For example, if the tag name of a tag is used as an argument, the task copies the value of that tag and sends it to the procedure as the argument.

The original value of the tag is not affected. Values modified as arguments cannot be passed back to the calling procedure.

An array cannot be used as an argument to a procedure, but an array tag can.

The declaration section of a procedure definition is optional. Any of the declarations can be made in this section. Remember the two previously stated rules:

- Any variables declared within the procedure are by definition local variables and cannot be referenced outside of the procedure.
- Declarations must come before any statements.

Calling Sequence—You must specify a procedure call using one of the following interchangeable forms:

```
{CALL} proc_name[(type1 arg1] [, type2 arg2...])]
{CALL} proc_name (type1 arg1 [, type2 arg2...])
```

where the keyword CALL is optional.

Developer-Defined Local Procedures

Calls to local procedures, with or without arguments, are normally made using one of the forms in the following table.

Syntax	Purpose
f( ) or f	To pass no arguments
f(arg1)	To pass arg1
f(arg1, arg2)	To pass arg1 and arg2

Procedure names can be up to 16 characters, must conform to the naming rules for variables, and can be followed by a set of parentheses containing the function's input parameters (arguments), if any are required.

Library Functions

Math and Logic has several predefined, specialized procedures, known as library functions. Expressions can include calls to library functions, which are grouped into five categories:

- Directory/Path Control
- Mathematical
- String Manipulation
- Programming Routines
- Miscellaneous Routines

The functions within each category are described in the following sections. Included in each function's description is a sample format of the function and an example of its use. Functions can vary among different operating systems. Refer to your operating system documentation for information about specific functions for a particular operating system.

Directory/Path Control Functions

Directory and path control functions are unique to each operating system.

Function	Sample Format	Description
getdir	string = getdir(drive)	Returns the current path of specified drive. drive = 0 current drive drive = 1 a: drive = 2 b:
	Example:	string = getdir(1)
	Therefore:	string = a:
getdrive	drive = getdrive	Returns current disk number. drive = 1 a: drive = 2 b:
	Example:	drive = getdrive
	Therefore:	drive = 2 (if current drive is drive **b**)
setdir	status = setdir(drive, path)	Returns zero for success. Sets new current drive/directory. drive = 0 current drive drive = 1 a: drive = 2 b:
	Example:	status = setdir(1,"a:\test")
	Therefore:	status = 0 (if current drive/directory successfully set to **a:\test**)
setdrive	stat = setdrive(drive)	Returns zero for success. Sets new current drive. drive = 1 a: drive = 2 b:
	Example:	stat = setdrive(1)
	Therefore:	stat = 0 (if current drive successfully set to drive **a**)

Mathematical

Function	Sample Format	Description	Example	Result
abs	**x** = abs(y)	Returns absolute value	x = abs(-5)	x = 5
cos	**x** = cos(y)	Returns the cosine of *y* (specified in radians)	x = cos(.4)	x = 0.921061
exp	**x** = exp(y)	Returns *ey*	x = exp(4)	x = 54.59815
log	**x** = log(y)	Returns the log base 10 of *y*	x = log(100)	x = 2
loge	x = loge(y)	Returns the natural log of *y*	x = loge(1)	x = 0
pow	x = y pow (z)	Returns *y* to the zth power	x = 2 pow (3)	x = 8
rnd	x = rnd	Returns a pseudo-random positive integer within the range 0 to 32767	x = rnd	x = 32750 (One possible result)
sin	x = sin(y)	Returns the sine of *y* (specified in radians)	x = sin(1.5)	x = 0.9974951
sqr	x = sqr(y)	Returns the square root of *y*	x= sqr(144)	x = 12
tan	x = tan(y)	Returns the tangent of *y* (specified in radians)	x = tan (.785)	x = 1

String Manipulation

Function	Sample Format	Description	Example	Result
alltrim	string = alltrim(string)	Returns *string* with leading and trailing blanks trimmed	msgvar = alltrim(“SMITH”)	msgvar = “SMITH”
asc	x = asc(string)	Returns the ASCII code for the first character in *string*	x = asc (“TEN”)	x = 84 (84 is the ASCII code for T)
chr	string = chr(var)	Returns equivalent character of an ASCII code	msgvar = chr(66)	msgvar = “B” (66 is the ASCII code for B)
instr	x = instr(str1, str2)	Returns the offset into *str1* of the occurrence of *str2*	x = instr(“ABCDE”,“B”)	x = 2
len	x = len (string)	Returns the length of *string*, not including the terminator	x = len(“MIAMI, FLORIDA”)	x = 13
lower	string = lower (string)	Returns *string* converted to lower case	msgvar = lower (“NOT”)	msgvar = “not”
ltrim	string = ltrim (string)	Returns *string* with leading blanks trimmed	msgvar = ltrim (“SMITH”)	msgvar = “SMITH”
substr	string= substr (string, offset, len)	Returns a string of *len* length or less beginning with the offset character from the beginning of *string*. The offset to the first character in *string* is 1.	msgvar = substr(“ABCDE”, 3, 2)	msgvar = “CD”
trim	string = trim(string)	Returns *string* with trailing blanks trimmed	msgvar = trim(“SMITH”)	msgvar = “SMITH”
upper	string = upper(string)	Returns the input, *string*, converted to upper case	msgvar = upper(“not”)	msgvar = “NOT”

Programming Routines

Syntax	Description
EXIT(status)	Exits the program and sets the program return status
CALL procname([p1...])	Calls a procedure. The keyword CALL is not required. See “Procedure Call Statements” on page 329.
INPUT string_prompt, var1, var2...	Accepts input from keyboard. The first field entered is placed in **var1**. The first comma entered begins the second field, which is placed in the **var2**, and so on.
LOCK	Locks the database. No other task can access the database while it is locked. A LOCK statement delimits a block of code to execute in critical mode, without interference from other FactoryLink tasks running on the system. Each LOCK statement must have an UNLOCK statement.
UNLOCK	Unlocks the database, allowing other tasks to access it. Must be issued for every LOCK. If time-consuming code is included between LOCK and UNLOCK statements, performance may be affected, because no other tasks can access the database while it remains locked.
PRINT “Row and line:”, row1, line	Sends each listed print parameter (variable) to the display, converting to ASCII, if necessary.
TRACE expr	While **expr** remains true, each line assignment and the procedure exit point print as they run. **Note:** TRACE is not supported in CML.

Miscellaneous Routines

Function	Sample Format	Description	Example	Result
argcnt	**x** = argcnt	Returns the number of program arguments	x = argcnt	x = 3 (if the number of program arguments is 3)
getarg	**x** = getarg(argnbr)	Returns the specified program argument	x = getarg(2)	x = TEST (if the second argument is the directory name, which is TEST)
system	**x** = system (“cmd/c call”)	Allows calls to the operating system. The command is formatted as a string. This might be a simple directive, such as DIR C:, or it might be complex, such as execute a batch file to request values or inputs from the user (“@GETID”) or to request another operating system shell (“DOSSHELL”). To transmit one backslash (\) character, type two backslashes. Doubling the character causes the system to recognize it as a literal character.	x = system(“cmd/c copy c:\\file.txt b:\\filebak.txt.”)	In Windows, the system function is asynchronous, so the system function returns immediately and the command runs in parallel. x = 0 (if the command started) x=2 (if the specified file was not found) x=3 (if the specified path was not found)

RUNNING PROGRAMS AS INTERPRETED

Start the application by typing the FLRUN command at the system prompt to run those programs that have Interpreted entered in the **Mode** field of the Math and Logic Triggers Information table as interpreted. Math and Logic begins executing interpreted programs by loading them into memory. After loading and validating the programs, Math and Logic waits for changes to the trigger tags in the real-time database associated with the procedures in the program. When a trigger tag is set to 1 (ON), the task executes the program associated with that trigger.

Each time an interpreted program is executed, Math and Logic first reads or interprets, the instructions within the program to determine the actions to perform, then it executes those actions.

CML PROCESS

CML contains utilities and libraries that are used along with a third-party ANSI C/C++ compiler to generate ANSI C code from the ***.prg** files you created. When you have completed configuring the Variables Table, Triggers Table, and Procedures Table, you have created the processing procedures for running programs in either IML or CML. The following discusses the process involved in producing an executable file for the given domain from the .PRG files.

The compile process begins at run time on a development system, when CML:

1. Translates the program (.PRG) files into C source code
2. Puts the C code into files with an extension of .C
3. Compiles the .C files to produce object (.obj) files
4. Links the object files to the appropriate libraries to create binary executable (.exe) files
5. Runs the executable file as each program's associated trigger(s) are set.

> **Note:** After the CML files have been tested and approved for use, the executable files can be copied to a run-time system that has the CML option enabled. A compiler is not needed on the run-time system.

Because FactoryLink applications can be configured in both Shared and User domains, CML creates one executable file for each domain that contains the .PRG files. The file name of each executable is unique. The filename begins with a C and is followed by the domain name:

- {FLAPP}/SHARED/CML/CSHARED.EXE for the Shared domain
- {FLAPP}/USER/CML/CUSER.EXE for the User domain

CML includes three utilities that create the executables CML used at run time:

- MKCML
- PARSECML
- CCCML

Each utility performs a specific role in the compile process as shown in the call sequence in Figure 14-2. Utilities are started in a specific order:

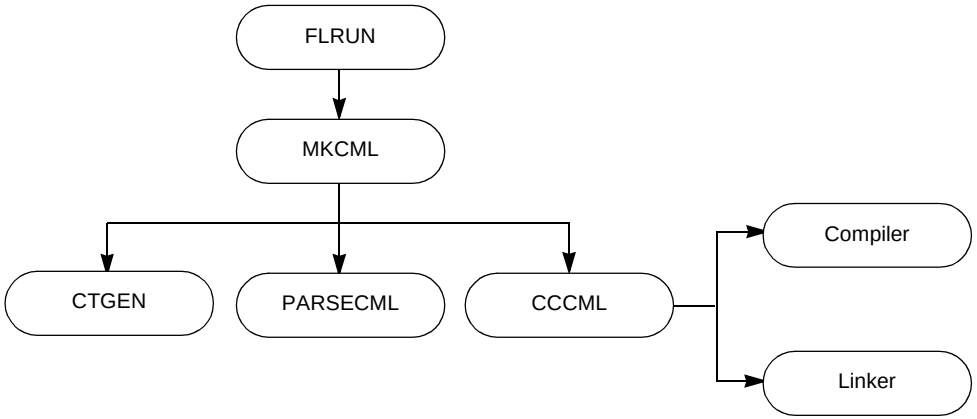
1. FLRUN calls the MKCML utility. The FLRUN command sets the FactoryLink path, the application directory path, the user name, and the domain name to the environment variables and turns off the verbose-level and clean-build parameters.

   Note: CTGEN (and GENDEF) run normally as part of FLRUN. If you are debugging and need to run the items separately, always run CTGEN and GENDEF before running MKCML. MKCML calls CTGEN, which ensures the Math and Logic .CT file is up to date.

2. MKCML calls PARSECML to produce .C files from the program (.PRG) files.

3. MKCML then calls CCCML to compile the .C files into object files using an external compiler. Using an object linker, the object files are linked with library files into binary executable files.

Figure 14-2 CML Utilities Call Hierarchy



MKCML

The MKCML utility is a shell that calls the PARSECML and CCCML utilities as needed for the current application. For each domain, MKCML checks the dependencies between the configuration tables (named IML.CT for both IML and CML) and the program files. MKCML performs these tasks:

- Calls CTGEN which compares IML.CT against the database files. If the database files have a later time/date stamp than IML.CT, CTGEN rebuilds IML.CT to bring it up to date.
- Determines whether the time/date of IML.CT has changed. If so, MKCML reproduces and recompiles all of the .C files by calling PARSECML and CCCML.

When you redirect the output of MKCML to a file, the messages displayed in the output appears out of order because of the method used by the operating system to buffer and output messages. If you do not redirect the output of MKCML, the messages are reported to the standard output in the correct order.

PARSECML

The PARSECML utility parses the application program files and produces .C files for each domain. It produces a .C file for each program file if the program Mode field is set to COMPILED in the Math and Logic Triggers Information table.

This utility also checks the dependencies between the program files and the .C files to determine if any procedures were updated since the .C files were last produced.

PARSECML has various levels of debugging via the -Vx parameter that can generate more detailed output or even add debugging statements to the C code.

CCCML

The CCCML utility compiles each .C file produced by PARSECML into an object file using an external compiler. It then links the object files with the FactoryLink and developer-supplied libraries into a binary executable. To determine the name of the compiler to use for a specific operating system, CCCML uses a special file called a makefile named: {FLINK}/CML/CML.MAK.

Its debugging levels provide minimal information; for example, the exact command line used to compile and link the code. The CML variables in Table 14-3 provide manipulation of the CML environment.

Table 14-3 Miscellaneous CML Commands

Variable	Definition	Example
CC	Designates the command line compiler	
CCFLAGS	Specifies the command line options for compiling the .C files	-dos2 -AL -Au -Od -Zp -G2s -nologo -c -I{FLINK}\inc
CMLOBJS	Stock CML files not application dependent (may change as versions change)	glvars.obj cmlprocs.obj
USEROBJS	Allows for inclusion of user-defined object module at link time	
LINK	Designates the command line linker	/NOE/ST:16384/se:512
CMLLKOBJS USERLKOBJS	Allows for inclusion of object modules not usually part of CML.	
CMLLIBS USERLIBS	Allows for inclusion of libraries not usually part of CML	CMLLIBS: {FLINK}\LIB\FLIB.LIB\ {FLINK}\LIB\CML.LIB
DEFFILE	Specifies the link definition file that contains information about window attributes, resources, or compiled output options. Do not edit this file.	{FLINK}\CML\CML.DEF
TARGET	Destination of the executable	{FLAPP}\{FLDOMAIN}\cm, l\c{FLDOMAIN}.exe

System and Domain Makefiles

A makefile is a file that contains the information the CCCML utility needs to compile the .C files produced by PARSECML and to create an executable for the current domain. CCCML uses a makefile named **cml.mak**, which is unique for each operating system.

The cml.mak file, located in the {FLINK}/CML directory, typically contains the following information to create the final executable file:

- Name of the C compiler to use for a given operating system
- Command-line switches to be used when compiling
- Name of the operating system's object linker
- Linker command-line switches
- References to the FactoryLink libraries to be linked
- References to the developer-supplied libraries to be linked

As an aid for advanced users, CML provides a method for editing the cml.mak file. You can change the compiler and linker options, specify command-line switches, and specify which object files and libraries to link, providing the flexibility to create a makefile unique to an application for a given domain.

CML provides two file options: System Makefile and Domain Makefile. Both files for these options must retain the same name: cml.mak.

Math and Logic System Makefile

The cml.mak file in the System Makefile folder sets the defaults to control the compile job instructions for CML procedures. Any changes made to this file are global; they apply to all applications on the system. However, it is not recommended that any changes be made to this file. Any definitions in the system-specific makefile in the application directory override the definitions in the master makefile in the /FLINK/CML directory.

If the cml.mak file requires editing, expand the **Math and Logic System Makefile** folder, open **cml.mak** (the same file from the {FLINK}/CML directory), edit the file as required, and then save the changes.

Math and Logic Domain Makefile

A domain-specific makefile does not exist until you create one. Once created, this makefile is used for the domain instead of the system makefile.

To create a domain makefile, either copy the cml.mak file from the {FLINK}/CML directory to the {FLAPP}/{FL DOMAIN}/CML directory for the Shared domain, or right-click the **Math and Logic Domain Makefile** folder and click **New**. A new file that is an exact copy of the system makefile is created. Edit the file as required, and save the changes. Any definitions in the domain-specific makefile in the application directory override the definitions in the master system makefile in the {FLINK}/CML directory.

Using Global Procedure Variables in CML

For Compiled Math and Logic, declare global procedure variables in either of two ways:

- Before the first procedure definition in a program file, as in Interpreted
- In an include file, which contains all of the global variables and procedures that would otherwise be declared at the top of each program file that referenced them.

For example, using any text editor, create and edit an include file with an .INC extension, containing the following text:

```
DECLARE PROC testproc
DECLARE SHORT _val1
DECLARE FLOAT _val2
```

Include files must have an .INC extension so the system can open and save them during an FLSAVE. Include files are located in the PROCS directory of the current domain and current application.

For example, the previous include file is saved to path:
FLAPP\FLDOMAIN\PROCS\MYPROG.INC

where

MYPROG.INC

Is a developer-defined file name.

Use the keyword **include** to declare the include file with any program file to be run in the compiled mode. The syntax is

include "MYPROG.INC"

The keyword **include** instructs Math and Logic to read the contents of the **include** file and include it as part of the current program file.

Note: **Include** causes a validation error even though it is evaluated properly at compile time. An alternative is to use the C include within a **cbegin cend** block. For example,

```
cbegin
#include <time.h>
cend
```

The following example shows how to use an **include** file (procedures **p1** and **testproc** with an include file):

```
Include file test.inc

DECLARE PROC testproc()
DECLARE SHORT _val1
DECLARE FLOAT _val2
```

```
Procedure p1

include “test.inc”

PROC p1
BEGIN

CALL testproc()

END
```

```
Procedure testproc

include “test.inc”

PROC testproc()
BEGIN

   _val1=0
   _val2=0.0

END
```

Using Global Procedure Variables in CML

LOCAL STRING variables declared inside the body of a procedure in CML get allocated on the stack when the procedure executes. Since the default stack size for the C++.NET compiler is 1 Megabyte, if there are numerous STRING variables or large arrays of STRING variables, the stack size could be exceeded and the program would abort with a stack overflow error message. If this occurs, it can be resolved by declaring the variables outside the body of the procedure so that they get allocated in global memory which doesn’t have the same limitation. You can also adjust the compile/link flags for the C++.NET compiler to increase the stack size, by adding the “/STACK:reserve[,commit]” link flag to make the file for CML. Refer to the compiler documentation for more details.

RUNNING CML

CML compiles and runs on both development systems and run-time systems.

CML on a Development System

Before starting the Run-Time Manager, **FLRUN** invokes several utilities to compile programs into a single executable file. The compiled programs will have **COMPILED** entered in the **Mode** field of the Math and Logic Triggers Information table.

CML on a Run-Time-Only System

The CML development system executables must be transferred from the development system to the run-time system to run CML on a run-time-only system. Perform the following steps to run CML on a run-time-only system:

1. Use either of the following methods to transfer the CML executables to the run-time system:
   - Use the FLSAVE and FLREST utilities to perform a save and restore of the application from the development system to the run-time system. This saves and restores the compiled CML task along with the rest of the application.
   - Copy the executables from {FLAPP}/USER/CML or {FLAPP}/SHARED/CML on the development system to the same path on the run-time system.
2. Start CML. Depending on whether the **R** flag was set in the System Configuration Information table, do one of the following:
   - If the **R** flag was set, right-click the application name and select **Start**.
   - If the **R** flag was not set, start CML from the Run-Time Manager (RTMON).

The compile process begins and CML creates the executables. Because the development and run-time operating systems are the same, CML runs as is.

Running Utilities from Command Prompt Window

CML is designed so each of the CML utilities can be started from the command prompt window. This is useful when only a portion of the compile process needs to be processed. Table 14-4 identifies the command line parameters used by all CML utilities.

Table 14-4 CML Command Line Parameters

Parameter	Description
-P	Sets the path to the FactoryLink program files
-A	Sets the path to the application directory
-U	Sets the user name
-N	Sets the domain name
-V*x*	Sets the verbose (debug) level to *x* where 1 is the lowest and 5 is the highest value
-C	Performs a clean build, reproducing all files from scratch

ADVANCED TECHNIQUES

Customizing the Procedure Header File

Customizing of header information for the .PRG files on the FactoryLink server is provided by editing a text file. The file location and name are {Flink}\MSG\MLProcHeader.txt.

The MLProcHeader.txt file can be edited in any standard text editor, such as Notepad. The MLProcHeader.txt is created after a user creates the first .PRG file on the server. The edits appear in all **new** .PRG files. As additional edits are made to MLProcHeader.txt, the new edits appear only in .PRG files created after the edit is made. Table 14-5 shows the tokens and values provided for the edit customizing.

Table 14-5 Math and Logic Tokens and Values

Token	Value
$FILENAME	File name including the extension (not the full path)
$PROCNAME	Procedure name
$REPORTNAME	Report name
$FILEPATH	Full path to the file
$FLAPP	FactoryLink application directory
$FLAPPDIR	FactoryLink application directory
$FLINK	FactoryLink server directory
$FLINKDIR	FactoryLink server directory
$DOMAIN	Domain
$DATE	Date (long format)
$TIME	Time (long format)
$LONGDATE	Date (long format)
$LONGTIME	Time (long format)
$SHORTDATE	Date (short format)
$SHORTTIME	Time (short format)
$MEDIUMDATE	Date (Medium format)
$MEDIUMTIME	Time (Medium format)
$GENERALDATE	Date (General format)

Customizing Date and Time Formats

The date and time formats provide further customizing as shown in Table 14-6.

Table 14-6 Formatting Descriptions for Math and Logic Tokens

Item	Format Description
General Date	Displays a date according to your system settings: If real numbers, a date and time display, such as 4/3/04 05:34 PM. If no fractional part, only a date displays, such as 4/3/04. If no integer part, only time displays, such as 05:34 PM.
Long Date	Displays a date according to your system's long date format.
Medium Date	Displays a date using the medium date format appropriate for your language version of Microsoft Outlook; for example, the English-U.S. version includes the day, month, and year (05-Jan-04).
Short Date	Displays a date using your system's short date format.
Long Time	Displays a time using your system's long time format, including hours, minutes, seconds.
Medium Time	Displays time in 12-hour format using hours and minutes and the AM/PM designator.
Short Time	Displays a time using the 24-hour format, such as 17:45.

Calling C Code

The Math and Logic program uses three CML-specific keywords to call C code: cfunc, cbegin, and cend. This functionality is very powerful and flexible, but should be used sparingly because it makes your system harder to maintain in the future.

Using cfunc

Use the keyword cfunc to declare standard C functions and user-defined C functions as callable in-line functions within a CML program. In-line C functions allow a CML program to call a C function directly without opening a C code block. The function must be declared before it is called.

The C code generated by CML provides prototypes for standard library functions; however, it does not include prototypes for user-defined C functions. You must provide function prototypes for all user-defined functions. Including a function without a prototype may result in compiler warnings regarding the missing functions.

Use only C functions that use the Math and Logic data types of SHORT, LONG, FLOAT, and STRING with cfunc. Although a C function may use any data type internally, its interface to Math and Logic must use only these types.

In the following example, testfunc is declared to use four arguments whose values are SHORT, LONG, FLOAT, and STRING data types and to return a value with a SHORT data type:

```
DECLARE cfunc SHORT testfunc(SHORT,LONG,FLOAT,STRING)
```

You may declare C functions to return the following data types:

Function:	Value returned:
SHORT	Short-integer
LONG	Long-integer
FLOAT	Floating-point
STRING	String
VOID	None

The VOID data type is unique to CML. Use **VOID** when declaring a function not required to return a value. Do not use **VOID** in programs designed to run in interpreted mode.

Example 1—uses cfunc to declare the standard C function strcmp( ) for use within a CML program:

```
DECLARE cfunc SHORT strcmp(STRING, STRING)
PROC TEST(STRING s1)
BEGIN
   IF strcmp(s1,"QUIT")=0 THEN
      PRINT “QUITTING\n”
   ENDIF
END
```

The function **strcmp( )** compares two strings and returns a value that indicates their relationship. In this program, **strcmp** compares the input string **s1** to the string **QUIT** and is declared to have a return value of the data type SHORT.

- If the return value equals 0, then **s1** is identical to **QUIT** and the program prints the message **QUITTING**.
- If the return value is less than or greater than 0, the program prints nothing.

C functions declared using cfunc have full data conversion wrapped around them, meaning any data type can be passed to and returned from them.

Given the previous sample code, the following program is legal within CML:

```
PROC MYPROC
BEGIN
DECLARE FLOAT   _f
DECLARE LONG    _k
DECLARE STRING  _buff
   _buff=strcmp(_f,_k)
END
```

In this program, **strcmp** converts the FLOAT value **f** and the LONG value **k** to strings, compares the two strings, and then returns a number (buff) that indicates whether the comparison was less than, greater than, or equal to zero. This comparison is:

- If f < k, then buff is a number less than 0.
- If f = k, then buff is equal to 0.
- If f > k, then buff is a number greater than 0.

Example 2—uses cfunc to declare the function testfunc which has a return data type of VOID:

```
DECLARE cfunc VOID testfunc(FLOAT)
PROC MYPROC
BEGIN
DECLARE FLOAT _flp
   _flp=100.0
   testfunc(_flp)
END
```

In this program, the declared floating-point variable **flp** is set to 100.0 and this value is passed to the function **testfunc**. Note that VOID is entered in place of the data type for the function's return value. This is because the program is only passing a value to **testfunc** and the function is not required to return a value.

Using cbegin and cend

You can use the keywords **cbegin** and **cend** to embed C code directly into a CML procedure. Between these keywords, you can call external library functions and manipulate structures and pointers Math and Logic does not support; however, you cannot declare C variables inside a **cbegin/cend** block already within the scope of a procedure. When you declare a C variable, the declaration block from **cbegin to cend** must be displayed outside the procedure, above the PROC statement. See the declaration of **static FILE *Fp=stderr** in Example 2.

The **cbegin** and **cend** statement must each be on a line by itself with no preceding tabs or spaces. All lines between these two keywords (the C code block) are passed directly to the .C file that PARSECML produces for this program.

The following examples show how to use the cbegin and cend keywords.

```
# Example 1:
PROC TEST(STRING message)
BEGIN
DECLARE STRING buff
   IF message="QUIT" THEN
      PRINT “FINISHED.\n”
   ENDIF
cbegin
   sprintf(buff,"The message was %s\n",message);
   fprintf(stderr,buff);
cend
END
```

In this program, the **sprintf** and **fprintf** functions, called between **cbegin** and **cend**, are passed directly to the .C file that PARSECML generates for TEST. Note that local variables are within the scope of the C code block and can be accessed during calls to external functions.

Any C code blocks outside the body of a CML program are collected and moved to the top of the generated .C file, as shown in Example 2. In this program file, the statement: **static FILE *Fp=stderr;** is moved to the top of the program file just after the line **include “mylib.h”**.

```
# Example 2:
cbegin
#include “mylib.h”
cend
PROC TEST(STRING s1)
BEGIN
   PRINT “The message is ”,s1
END
cbegin
static FILE *Fp=stderr;
cend
PROC SOMETHING (FLOAT f1)
BEGIN
cbegin
   fprintf(Fp,"%6.2g\n",f1);
cend
END
```

The following example shows how to access tags from within embedded C code blocks. It increments the values of two analog tags, Tag1 and Tag2[5], by 10. Notice, the variable **task_id** is a predefined global CML variable and does not need to be declared.

```
PROC example
BEGIN
cbegin
{
   TAG tag[2];
   ANA value[2];
   fl_tagname_to_id(tag,2, “TAG1”,“TAG2[5]”);
   fl_read(Task_id,tag,2,value);
   value[0] += 10;
   value[1] += 10;
   fl_write(Task_id,tag,2,value);
}
cend
END
```

The following example shows how to manipulate message tags within embedded C code (**cbegin/cend** code blocks). This example reads from TAG1, adds X to the string, then writes the result to TAG2.

```
PROC ADD_X
BEGIN
cbegin
{
   #define MAX_LEN 80  /* default maximum message length */

   TAG tags[2];
   FLMSG tag1, tag2;
   char string_buff[MAX_LEN+1]; /* max length plus terminating 0 */

   tag1.m_ptr=tag2.m_ptr=string_buf;
   tag1.m_max=tag2.m_max=MAX_LEN;
   fl_tagname_to_id(tags,2,TAG1,TAG2);
   fl_read(Task_id,&tags[0],1,&tag1);
   strcat(string_buf,X);
   tag2.m_len=strien(string_buf);
   fl_write(Task_id,&tags[1],1,&tag2);
}
cend
END
```

When values are assigned to and read from MESSAGE tags in the normal syntax for the procedure files the **MAX LEN** field is limited to 1023 characters. All message values are truncated at 1023 characters. The function **fl_write ( )** must be called directly to store values longer than 1023 characters into a MESSAGE tag. The following example shows how to use a C macro to call the procedure msgtest to store a 90-character constant into the MESSAGE tag **msgtag**:

```
MSGTEST.PRG
cbegin
#define assign_msg(tagname, value) {\
   TAG tag; \
   FLMSG msg; \
   char buf[] = value; \
   fl_tagname_to_id(&tag,1,tagname); \
   msg.m_ptr = buf; \
   msg.m_len = strlen(buf); \
   msg.m_max = strlen(buf)+100; /* leave plenty of room */ \
   fl_write(Task_id,&tag,1,&msg); \
}
cend

PROC msgtest
BEGIN
   cbegin
   assign_msg(msgtag,123456789012345678901234567890123456789012345
   67890123456789012345678901234567890123456789012345678901234567890)
   cend
END
```

Calling Functions that Operate on Tag IDs

CML provides a function that looks up tag names and retrieves their tag IDs so developers can call and use functions that operate on tag IDs and not tag names. This function is

```
int fl_tagname_to_id(TAG*tp, int num, char*tagname,...);
```

where

TAG*tp	Is a pointer to a developer-supplied tag array to be filled in with tag IDs
int num	Is the number of tag names to look up.
char*	Is one or more character pointers to valid tag names.

This function returns a code indicating either GOOD or ERROR. It is designed for developers who integrate C source code into their Math and Logic programs and is available through the CML run-time library.

By using **fl_tagname_to_id( )** inside CML C code blocks, developers can look up one or more tag names and fill in a developer-supplied tag array with the tag ID for each tag name. Developers can then use these Tag IDs with the FactoryLink PAK functions, and any other function that operates on the tag ID instead of the tag name, just as the Math and Logic grammar does.

fl_tagname_to_id( ) is a variable argument function like print. The developer can retrieve as many valid tag IDs as tag array has room for.

The following example shows how to use **fl_tagname_to_id( )**:

```
cbegin
void   myfunc()
{
TAG list[2]
   fl_tagname_to_id(list, 2, “TIME”, “DATE”);
.
}
cend
```

In this example, the function retrieves the tag IDs for the two tags TIME and DATE and places their IDs into the tag array named list.

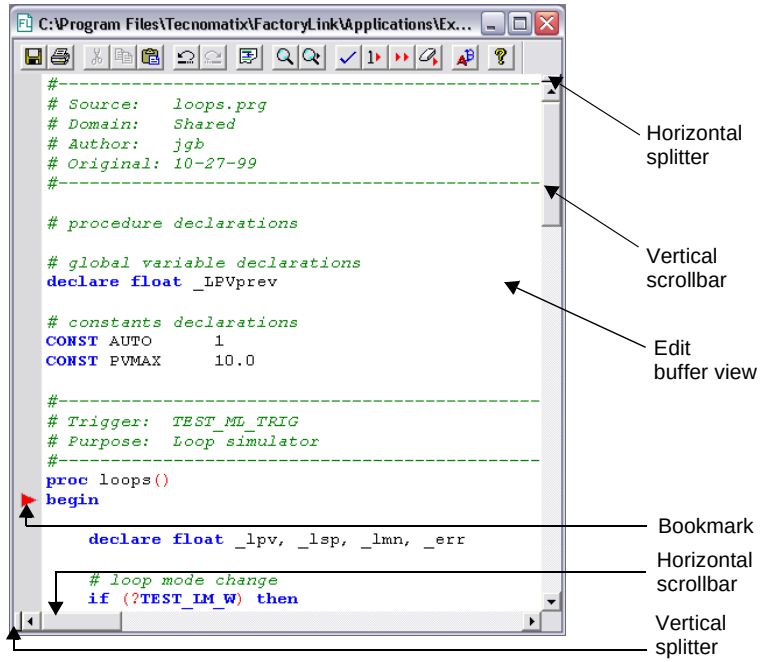
MATH AND LOGIC EDITOR

The Math and Logic editor is a Single Document Interface (SDI) application that allows multiple concurrent instances to run in the FactoryLink environment. This editor is used to create and modify the program files and associated procedures using the C-type programming language standards. If another editor is used, it is important that you validate the procedures you write with the Math and Logic editor.

The Math and Logic editor supports these features:

- Configurable window properties for customizable views
- Split horizontal and vertical views of the same edit buffer which can be scrolled separately
- Numerous edit commands with keystroke functions enabling, for example, quick copy-paste, drag and drop, find-replace, tab-to-space/space-to-tab conversion
- Chroma coding to assist with the identification of keywords, operators, declarations, global tags, commented lines, and strings.
- Keyboard shortcuts

Figure 14-3 Math and Logic Editor Screen



For information about these functions, see the *Configuration Explorer Help*.

SYSTEM CONFIGURATION TABLE SETTINGS

To view the Math and Logic task's system configuration settings, open **System > System Configuration > System Configuration Information** folder. Double-click **Interpreted Math and Logic**. The System Configuration Information dialog box appears.

To enable CML processing, the CML task must be added anywhere in the task list. The IML task does not need to be removed. Both tasks can be enabled and run at the same time. Adding a task requires displaying an existing task to use the dialog box as a template. The new task is added to the list below the displayed task. The position of the task in the list does not determine its rank in the run-time process; the **Start Order** field determines the run-time rank.

To add a task, double-click an existing task in the list, such as the Interpreted Math and Logic task. In the System Configuration Task dialog box, click the arrow-asterisk button at the bottom of the dialog box. Complete all the fields using the information in Table 14-7. Click **Apply** to complete the task. Refresh the application tree to display the new task in the list.

For more information about adding and modifying task parameters, see the *Configuration Explorer Help*.

Table 14-7 IML and CML Fields

Field Name	Definition		Default Field Data	
			IML	CML
Domain	Current FactoryLink versions typically use the Shared domain, but both are available.		Shared	Shared
Task Information	Task Name	Predefined name for the task, which cannot be changed	IML	CML
	Task Description	Optional alphanumeric description	Interpreted Math and Logic	(optional) Compiled Math and Logic

Table 14-7 IML and CML Fields (continued)

Field Name	Definition		Default Field Data	
			IML	CML
Task Flags	Run at Startup	**R**: Invokes task at FactoryLink startup. Must set to use IML or CLM.	optional	optional
	Create Session Window	**S**: Provides the process with its own tab window. Output prints to the Configuration Explorer Output window.	optional	optional
	Suppress Online Configuration	**O**: Suppress online updates for this process	optional	optional
	Suppress Task Hibernation	**H**: Not applicable for Math and Logic functions	not applicable	not applicable
Flag String Value	Input box	Displays value code of selected Task Flags: **F**: Foreground Flag. Puts this task in the foreground at startup	F	not applicable
	Edit Flags Directly	If selected, allows user input of string values to input box		
Task Options	Start Order	Specifies run-time rank for invoking the task when FactoryLink is started. (The start order is coordinated with the function requirements in the procedures. If the Math and Logic procedures use the information from another process, that process must start first.	3	3
	Start Priority	Processing priority	201	201
Task Executable	Executable File	Path and name of the file that executes this task. CML	bin/iml	bin/cml
	Program Arguments	Codes that add customization to the functions of the task. See “Program Arguments” on page 364 for more information.	none	none

PROGRAM ARGUMENTS

Task	Argument	Description
Mkcml	-A<appdir> or -a<appdir>	Sets the path to the application directory
	-C or -c	Performs a clean build, reproducing all files from scratch
	-N or -n	Sets the domain name
	-P	Sets the path to the FactoryLink program directory
	-U	Sets the user name
	-V<#> or -v<#>	Sets the verbose (debug) level. (# = 1 to 5). The use of the verbose levels -V3 and above can cause the linker error, DGROUP Exceeds 64K. This is because of the large number of strings inserted in the printf( ) statements. Limit the use of -V3 and above to debug use only.
Math and Logic (Interpreted mode only)	-L	Enables logging of debug information to a log file
	-V#	Sets verbose level (# = 0 to 4)

Verbose-Level Parameters

When you use a verbose-level parameter, the utility displays messages about its progress as it performs its part of the compile process. This serves as a debugging aid. Table 14-8 shows the messages produced by each utility at the verbose level indicated.

Table 14-8 Verbose Setting Messages by Level

Utility	Verbose Level	Result Displayed	Run-time Effects
MKCML	1 or higher	Application name and domains as they are processed	None
CCCML	1 or higher	• Message, “Not authorized to run Math and Logic” if the system cannot find the run-time bit • Current application and domain being processed • Message, “No .PRG files are configured as COMPILED” • Message echoing the command line that calls the compiler or linker before making the call • Names of each file as it is compiled • Message indicating all files are up to date	None
PARSCML	1	Name of each .C file name as it is produced	None
	2	Verbose level 1 message, plus: • Comments, containing the original source lines of the Math and Logic program, placed by the utility at the start of the generated C code • All programs as they are parsed	None
	3 or higher	Verbose Level 1 and 2 messages	Prints statement upon entry and exit procedure
	4	Verbose Level 1, 2	Prints each line as it executes

ERROR MESSAGES

Math and Logic maintains a log file for IML error messages issued during FactoryLink execution. A copy of this log file resides in a log subdirectory under the Shared and/or the User domain directory associated with your FLAPP. Use any ASCII text editor to view the log file.

The following Math and Logic error messages can display on the Run-Time Manager screen, depending on the mode (IML or CML). Math and Logic configuration table files are named IML.CT regardless of the mode used (IML or CML).

Error Message	Mode	Cause and Action
Array access out of range	IML, CML	Cause: An array index has become too large or too small (probably inside a loop where it is repeatedly incremented or decremented) and is now indexing outside the defined boundaries of the array. Action: Edit the procedure so the specified index is within the range of the declared array.
Assignment to Unknown Type	CML	Cause: An assignment statement encountered an unknown type of data. Action: Verify declarations and assignment statements.
Bad tag array index	CML	Cause: You specified an index not within the bounds of an array, or made reference to a tag as an array not defined as an array. Action: Ensure the tags to be arrays are defined as arrays and have the proper indices.
Can't get FactoryLink ID = number	CML	Cause: Either the FactoryLink real-time database is not running or the task name is not listed correctly in the System Configuration table. Action: Correct the task name in the System Configuration table or start from the Run-Time Manager.
Can't locate MATH keyword file	IML, CML	Cause: The IMLTOKEN.KEY file, normally found in the /FLINK/KEY directory, does not exist or cannot be opened. The installation program may not have completed successfully, or the file may be damaged. Action: If IMLTOKEN.KEY does not exist, back up any applications, then reinstall FactoryLink.

Error Message	Mode	Cause and Action
Can't open Compiled Math and Logic makefile: filename	CML	Cause: The MAKE file is missing. Action: Reinstall the FactoryLink software.
CML executable missing	CML	Cause: A compilation or a link error a syntax error caused in one of the .PRG files occurred. Action: Check the .PRG files for syntax errors.
Compiled Math and Logic is not authorized within the software protection key	CML	Cause: The CML option was not purchased. Action: Purchase the CML option or use IML.
Compiled Math and Logic link module missing	CML	Cause: A module named in the MAKE file does not exist. Action: Remove the references to the nonexistent module.
Compiled Math and Logic source file .c file is missing	CML	Cause: A file named in the MAKE file does not exist. Action: Remove the references to the nonexistent module.
Corrupt configuration table index	IML, CML	Cause: The Math and Logic .CT files are damaged. Action: Delete the files. Restart to rebuild the .CT files.
Corrupt data in configuration file	CML	Cause: The Math and Logic .CT files are damaged. Action: Delete the files. Restart to rebuild the .CT files.
CT filename size incorrect. Struct size CT size	CML	Cause: The Math and Logic .CT files are damaged. Action: Delete the files. Restart to rebuild the .CT files.
CTGEN returned a FAILURE status - exiting MKCML	CML	Cause: CTGEN returned an error during compilation or linking to an executable. Action: Check the .PRG files and run MKCML again.
Database read error	IML, CML	Cause: An error occurred while Math and Logic was reading the tag from the FactoryLink real-time database. Action: Verify that FactoryLink is installed and set up properly by running the fltest application. If fltest is operating properly, check your configuration for errors.
Database write error	IML, CML	Cause: An error occurred while Math and Logic was writing to a tag in the FactoryLink real-time database. Action: Verify that FactoryLink is installed and set up properly by running the fltest application. If fltest is operating properly, check your configuration for errors.

Error Message	Mode	Cause and Action
Division by zero	IML, CML	Cause: The program encountered an attempt to divide a value by zero or to perform an operation directly or indirectly resulting in division by zero with the result undefined. Action: Edit the procedure to divide by a value other than zero. **Note:** Usually, this is an inadvertent result of an array or loop index being decremented out of control or the result of not testing operator input used as a divisor.
Environment table full - unable to add variable	CML	Cause: Not enough space exists in the operating system shell to set the environment variables. Action: Increase the environment space in the operating system shell. Refer to the documentation for the operating system for information.
EOF in codef	IML, CML	Cause: The task found an end of file character in the code array for this program before one was expected (for example, before the logical end of a program). Action: Correct your code to remove the EOF character.
Error occurred compiling file: filename	CML	Cause: The named file caused an error in the compile process. Action: Check the file for syntax errors.
Error occurred creating linker response file	CML	Cause: Memory allocation error. Action: Check the amount of memory.
Error occurred spawning application's CML executable	CML	Cause: An error occurred in the compile process that prevented the creation of the CML executable. Action: Check /FLAPP/SHARED/CML/CSHARED.EXE and /FLAPP/USER/CML/CUSER.EXE. Type the following command to rebuild the CML executables: `MKCML-C`
Error occurred writing to linker response file	CML	Cause: Memory allocation error. Action: Check the amount of memory.
Error opening generation output file(s)	CML	Cause: An error occurred while attempting to convert .PRG files to .C files. Action: Ensure the path name is valid and enough disk space is on the specified drive.

Error Message	Mode	Cause and Action
Error reading configuration table header	CML	Cause: The Math and Logic .CT files are damaged. Action: Delete the files. Restart to rebuild the .CT files.
Error reading procedure config. record	IML, CML	Cause: The Math Logic .CT files are damaged. Action: Delete the files. Restart to rebuild the .CT files.
Error reading tag config. record	IML, CML	Cause: The Math and Logic .CT files are damaged. Action: Delete the files. Restart to rebuild the .CT files.
Error reading trigger config. record	IML, CML	Cause: The Math and Logic .CT files are damaged. Action: Delete the files. Restart to rebuild the .CT files.
Error while reading DOMAIN.CT	CML	Cause: The DOMAIN.CT file in /FLAPP/FLDOMAIN/CT directory is damaged. Action: Delete the files. Restart to rebuild the .CT files.
Executable filename not found - exiting MKCML	CML	Cause: The system was started with the FLRUN command, which calls MKCML. MKCML could not find the executable CML.EXE because it was not created. Action: Use the MKCML utility to create CML.EXE
Executable filename returned a FAILURE status - exiting MKCML	CML	Cause: The creation of the CML.EXE aborted because a program file has errors. Action: Check the .PRG files so they validate with no errors from MKCML.
FactoryLink environment variables are not set properly	CML	Cause: You did not set the FLINK and/or FLAPP environment variables. Action: Set FLINK to the full path of the FactoryLink program files; set FLAPP to the full path of the application files.
Float domain error in function name arg1, arg2	IML, CML	Cause: The specified argument to a floating-point operation is not within acceptable range. Action: IML: Verify the argument in the operation. CML: Refer to the Math Library Functions in the documentation supplied with your compiler.
Float exception in procedure name arg1, arg2	IML	Cause: A problem occurred in a floating-point operation in the specified procedure. Action: Correct the problem in the function.

Error Message	Mode	Cause and Action
Float overflow in function name arg1, arg2	IML, CML	Cause: The specified value was greater than the maximum. (Often caused by inadvertent division by zero.) Action: IML: Correct the problem in the function. CML: Refer to the Math Library Functions in the documentation supplied with your compiler.
Float singularity in function name arg1, arg2	IML, CML	Cause: A problem occurred in a floating-point operation in the specified procedure. Action: IML: Correct the problem in the function. CML: Refer to the Math Library Functions in the documentation supplied with your compiler.
Float underflow in function name arg1, arg2	IML, CML	Cause: The specified value was less than the minimum allowable (representable) floating-point value. Action: IML: Correct the problem in the function. CML: Refer to the Math Library Functions in the documentation supplied with your compiler.
Invalid argument: arg	CML	Cause: An incorrect command line switch is used with the MKCML utility command. Action: The following command line switches are valid with the MKCML utility command: P, A, U, N, Vx, and C.
Invalid data type	IML, CML	Cause: The operating system, third-party software, or hardware setup on your system is incorrect or incompatible or an internal error occurred. Action: Verify that FactoryLink is installed and set up properly by running the fltest application. If it is operating properly, check your configuration for errors.
Invalid/missing domain specified	CML	Cause: The specified domain is invalid. Action: Correct the domain.
Invalid stack variable (Mode: IML and CML) Invalid symbol type	IML	Cause: The operating system, third-party software, or hardware setup on your system is incorrect or incompatible or an internal error occurred. Action: Verify that FactoryLink is installed and set up properly by running the fltest application. If it is operating properly, check your configuration for errors.

Error Message	Mode	Cause and Action
Invalid tag dimensions in config. record	CML	Cause: Specified an index not within the bounds of an array, or made reference to a tag as an array not defined as an array. Action: Ensure tags to be arrays are defined as arrays.
Invalid tag type	IML, CML	Cause: A variable is entered as a message or mailbox variable type. Action: Change the variable type to one of the four allowed types: SHORT, LONG, FLOAT, or STRING.
Invalid token in code array	IML, CML	Cause: An unrecognized instruction is found in the code array for this procedure. Action: Verify your procedure and try again.
Invalid trigger	IML	Cause: Math and Logic encountered an invalid trigger. Action: Verify your triggers and correct any errors.
Invalid variable block size	IML	Cause: Math and Logic encountered an invalid variable block size. Action: Verify your procedures and correct any errors.
Invalid variable type	IML, CML	Cause: Math and Logic encountered an invalid variable type. Action: Verify your procedures and tags. Correct any errors.
Keyword keyword used as constant	IML, CML	Cause: The named keyword is used as the name of a variable or constant in a procedure. Action: Substitute another name for this variable or constant within the procedure.
Length of line too long	CML	Cause: A program line is too long to process. Action: Break up the line into multiple lines.
Local index < 0	IML	Cause: The operating system, third-party software, or hardware setup on your system is incorrect or incompatible or an internal error occurred. Action: Verify that FactoryLink is installed and set up properly by running the fltest application. If it is operating properly, check your configuration for errors.

Error Message	Mode	Cause and Action
Lookup table is corrupt	IML	Cause: The task cannot find procedures because the table pointing to them is corrupted. Action: End the run and restart it.
Missing or invalid keyword: keyword	CML	Cause: The MAKE file contains an invalid keyword. Action: Correct the MAKE file.
Missing procedure: procedure name	IML	Cause: A program file exists with no procedure defined with the same name as the file or the current procedure attempted to call another procedure not defined or is not declared in the current program file. Action: Verify all program files contain a main procedure with the same name as the program file. Verify the called procedure is declared in the current program file.
filename missing CML makefile keyword: keyword	CML	Cause: The MAKE file contains an invalid keyword. Action: Action: Correct the MAKE file.
MOD by zero	IML, CML	Cause: The procedure attempted to perform a modulo operation resulting in division by zero with the result undefined. Action: Edit the procedure to use a value other than zero in the operation. Look for loop indices or accumulators used as divisors.
Multiple trigger entries	IML	Cause: Multiple triggers are defined for the same Math and Logic procedure or the trigger files are corrupt. Action: Correct the errors in the trigger file and restart.
Nesting too deep	IML, CML	Cause: The nesting of the local procedure is too deep. The maximum depth for local procedure nesting is 32, unless limited by available memory constraints. Action: Restructure the nesting of the procedure within the program file.
Nesting too shallow	IML	Cause: This usually indicates incorrectly nested logical structures, such as overlapping WHILE...WEND or IF...ENDIF statements. Action: Verify nested structures and correct any overlap errors.

Error Message	Mode	Cause and Action
No configuration tables defined	IML, CML	Cause: The Math and Logic .CT files in the /FLINK/CT directory have been damaged or no configuration table defined. Action: Delete the file if it exists. Restart the application to rebuild the file.
No tables configured for this task	IML, CML	Cause: The Math and Logic .CT files in the /FLINK/CT directory does not exist or cannot be opened. The installation may not have completed successfully or the file may have been damaged. Action: Delete the file if it exists. Restart the application to rebuild the file.
No triggers are defined for this task	IML, CML	Cause: No triggers are defined for the Math and Logic task. Action: Define one or more triggers in the Math and Logic Triggers Information table.
No variable name separator character =	CML	Cause: A syntax error is encountered in the MAKE file. Action: Correct the error and compile again.
Null procedure: procedure name	IML, CML	Cause: The current procedure attempted to call another procedure not defined or is in a program file that does not have a trigger defined in the Math and Logic Triggers Information table. Action: Verify the appropriate program file has a trigger defined.
Options not installed for Compiled Math and Logic	CML	Cause: The task was not be loaded because the configuration sequence or authorization code was entered incorrectly, or the CML option was not purchased. Action: Use the License Wizard to verify that the CML option is present.
Out of RAM	IML, CML	Cause: No more memory is available for the system to allocate to the task. Action: Close any unnecessary windows or programs. Add memory to the system if this error occurs often.

Error Message	Mode	Cause and Action
Out of RAM keywords	IML, CML	Cause: Insufficient memory is available to run the Math and Logic task. Action: Make more memory available by removing unnecessary software modifying the hardware.
Parsing error: see procedure name	IML	Cause: A parsing error occurred during program execution. Action: Review the Math and Logic message log file for details about the error and take corrective action.
Partial loss of significance in procedure name arg1, arg2	IML, CML	Cause: A problem occurred with numeric representation causing loss of significance in a numeric value during execution of the specified procedure. Action: IML: Correct the problem in the function. CML: Refer to the Math Library Functions in the documentation supplied with your compiler.
Procedure procedure name does not exist	IML, CML	Cause: A program file exists with no procedure defined with the same name as the file. Action: Verify all program files contain a main procedure with the same name as the program file. Ensure the case (upper, lower, or mixed-case) of the file name agrees with the case of the name in the file.
Procedure multiply defined	IML, CML	Cause: Two procedures in the same or different program files are defined with the same name (remember: file names in Math and Logic are case-sensitive) or a variable, constant, or tag is defined with a name already used. Action: Change the name of one of these procedures or variables so no two procedures have the same name.
Record size error	IML	Cause: An array may not be properly sized. Action: Check arrays to be sure that they are sized properly.
Run-time error in proc procedure name, Line line number: message	IML	Cause: You have an error in the indicated line number. Action: Correct the problem and run again.
Run-time error in proc procedure name: message	CML	Cause: You have an error in the indicated procedure. Action: Correct the problem and run again.
Stack overflow	IML, CML	Cause: You may have too many nested blocks in your code. Action: Rewrite your code to avoid excessive nested blocks.

Error Message	Mode	Cause and Action
Stack underflow	IML, CML	Cause: The operating system, third-party software, or hardware setup on your system is incorrect or incompatible or an internal error occurred. Action: Verify that FactoryLink is installed and set up properly by running the fltest application. If fltest is operating properly, check your configuration for errors.
Tag array mismatch	CML	Cause: The array is defined to have *x* dimensions, but is referenced as having more or fewer dimensions than defined. Action: Verify the array dimensions for the tag and reference the array accordingly.
Tag type does not match	IML	Cause: A tag name is declared in a Math and Logic procedure with a type different from the defined tag name. Action: Change the type of the defined tag name. This requires deleting the tag and all occurrences of that tag and then redefining the tag with the correct type.
Too many global variables	IML, CML	Cause: Too many global variables are declared in the specified procedure. Action: Remove or combine some of the variables.
Too many local variables	IML	Cause: Too many local variables are declared in the specified procedure. Action: Declare some variables outside of the procedures.
TOS overflow	IML, CML	Cause: The operating system, third-party software, or hardware setup on your system is incorrect or incompatible or an internal error occurred. Action: Verify that FactoryLink is installed and set up properly by running the fltest application. If it is operating properly, check your configuration for errors.
Total loss of significance in procedure name arg1, arg2	IML	Cause: A problem occurred with significance (proper representation of the entire value) in a numerical operation in the specified procedure. Action: IML: Correct the problem with the function. CML: Refer to the Math Library Functions in the documentation supplied with your compiler.

Error Message	Mode	Cause and Action
Trigger size error	IML	Cause: The operating system, third-party software, or hardware setup on your system is incorrect or incompatible or an internal error occurred. Action: Verify that FactoryLink is installed and set up properly by running the fltest application. If fltest is operating properly, check your configuration for errors.
Unable to set current working directory to: directory name	CML	Cause: Invalid directory specified. Action: Specify the directory set in the FLAPP environment variable.
Undefined identifier	IML	Cause: Invalid identifier specified. Action: Validate your code and correct errors.
Unknown tag type	CML	Cause: The tag type specified is unknown. Action: Validate your code and correct errors.
Wrong type	IML, CML	Cause: You declared a local or global variable with a data type invalid for the operation. Action: Edit the variable declaration to contain a valid data type.

Chapter 15 . . . .
Persistence

The Persistence task saves values from an active FactoryLink application at predetermined times to prevent loss of useful data if FactoryLink shuts down unexpectedly. These saved values are written to disk and are not affected when FactoryLink shuts down. Then, when you restart FactoryLink with the warm start command-line option, the Run-Time Manager restores the real-time database from the values in the disk file.

The memory-resident real-time database is a collection of tag values and it represents the current state of the application. The values of the tags are lost when the application is shut down because the real-time database is removed from RAM. When the application is started again, the real-time database is recreated and its tags are initialized to zero or their default values, if defined. This can be a problem if FactoryLink unexpectedly shuts down because of an event, such as a power loss or a faulty process. Useful information can be lost if it has not been saved. Persistence provides a way of saving the state of an active FactoryLink application.

Persistence is the ability of a tag to maintain its value over an indefinite period of time. Non-persistent tags lose their value when the Run-Time Manager exits and shuts down the real-time database. The Persistence task writes tag values to disk, making these tags persistent. The file the task creates is called a *persistence save file*.

The task offers the following features:

- Offers two ways to configure which tags will be persistent:
  - On a per-tag basis
  - On a domain basis
- Allows you to specify either a warm or cold start when starting FactoryLink:
  - Cold start initializes all tags to their default values as configured in the Tag Editor.
  - Warm start initializes all non-persistent tags to their default values just like a cold start and restores all persistent tags to their previously saved values.
- Uses its own internal disk cache to increase speed and reduce disk I/O overhead
- Resolves configuration changes to the application
- Allows you to specify a trigger tag that triggers the Persistence task to copy the current persistent save file to a backup file

OPERATING PRINCIPLES

Before configuring Persistence, you must first determine which tags must be saved, when their values are saved, and how these saved values are restored during a warm start. Then, specify this information in the Tag Editor for each persistent tag.

At run time, the Persistence task saves the values of the persistent tags to its own internal disk cache and then writes the data to disk from there. Saving the persistent values to memory first increases processing speed and ensures all values meant to be saved are saved within the allotted time.

The RESOLVE program, executed by the FLRUN command, creates a blank persistence save file the first time it is executed. At startup, the Persistence task loads the persistence save file to determine which tags in the application are persistent and when the values of those tags are to be saved. It also loads the PERSIST.CT file to get specific information about the configuration of the Persistence task itself.

Performing a Warm Start

Unless you tell FactoryLink to use the persistence save file, it will initialize all values in the real-time database with either zeros or default values, as usual. To initialize the database with values from the save file, you must use a special program argument, *-w,* when starting FactoryLink. The -w means *warm start,* and it tells FactoryLink to use the save file. When you warm start FactoryLink, Persistence loads the save file. Then the Run-Time Manager restores the real-time database from the values in the save file. Non-persistent values (those not in the save file) are initialized to either zero or their default values.

The **-w** command is already set for the Examples Application and Starter Templates. To add the **-w** command to another FactoryLink application, follow these steps:

1. Right-click the application in Configuration Explorer.
2. Click the field next to **FLRunArgs** and add **-w**. Be sure a space is between the last character in the command line and the dash in **-w**. Click **OK**.

Resolving Configuration Changes

Configuration changes to an application could affect the Persistence save file. After you reconfigure an application using Configuration Explorer, the tags and their values stored in the Persistence save file either may not exist or may not have the same data type when you restart the application. Before each FactoryLink session is restarted, the tag names stored in the Persistence save file must be checked for changes against the OBJECT.CT and the DOMAIN.CT files.

The RESOLVE.EXE program automatically resolves any configuration changes. The FLRUN command automatically executes this program before it starts the Run-Time Manager.

The RESOLVE program serves three purposes:

1. Creates the blank Persistence save file the first time it is run
2. Manages the changes between the Persistence save file and the FactoryLink configuration files
3. Determines if the Persistence save file is usable and, if not, the program looks for and uses the Persistence backup file

RESOLVE makes the following changes:

- Removes tag names from the Persistence save file that have been deleted from the application or changed to a different data type
- Updates the tag name ID for tag names that were deleted, then recreated
- Adds tag names to the Persistence save file that have been reconfigured to have Persistence (these tag names are added with no data values.)
- Copies the Persistence backup file over the Persistence save file if the save file is corrupted

CONFIGURING PERSISTENCE

Before configuring Persistence, you must first consider which tags in the application are critical to application startup and must be saved. This subset of tags from your application will be the ones you mark as persistent. It is not feasible for Persistence to save every tag in an application, so make sure that Persistence saves only those values that need to be maintained after the application shuts down. To make use of this save file after FactoryLink has shut down, you must restart FactoryLink with the -*w* argument.

There are two steps in configuring Persistence in an application:

1 Mark the tags to be saved as persistent.

2 Configure the Persistence task itself by completing the Persistence Save Information table.

3 Add the R flag to the Persistence task in the System Configuration table.

Marking the tags tells the Persistence task which tags to save, but the task does not run until you configure its table and set the R flag.

Marking Tags to be Persistent

Individual tags or an entire domain can be marked as persistent. Mark individual tags as persistent in the Tag Editor. Mark tags on a domain basis using both the Tag Editor and the Domain List. Each method is explained in the following sections. In both methods, you indicate how tag values will be saved, and how the tags' change-status bits will be set when their values are restored during a warm start of FactoryLink.

Persistence for Individual Tags

Configure Persistence for individual tags using the Tag Editor, which appears when you:

- Define a new tag in Configuration Explorer, or
- Press **Ctrl+T** in a **Tag** field for a previously defined tag.

The Persistence section of the Tag Editor is explained below:

Use Domain Settings	Saves the value of this persistent tag according to the option chosen in the Domain List. The Saving and Restoring options are disabled when this option is chosen. Clear **Use Domain Settings** to enable the Saving and Restoring options for this tag specifically.
Save	Indicates when the value of this persistent tag is saved. Click one, or both, of the following: On Time—Saves the value of the tag on a timed trigger. On Exception—Saves the value of the tag whenever its value changes.
When Restoring	Indicates how to set this tag's change-status bits when it's value is restored in the real-time database. Click one of the following: Set Change Status ON—Restores the tag with its change-status bits set to 1 (ON) after a warm start. Set Change Status OFF—Restores the tag with its change-status bits set to 0 (OFF) after a warm start. This is the default. For example, you may have several Math & Logic procedures triggered by digital tags but the application controls when these tags are force-written to a 1 (value = 1; change-status bits = 1). If you perform a warm start with Change Bits ON, all of the digital tags change-status bits are written to a 1 and all of your IML procs run at once.
No Options Selected	This tag is not marked as persistent.

To configure Persistence for individual tags, follow these steps:

1. Make sure the **Use Domain Settings** box is not selected.
2. Choose when you want to save this tag's value by clicking **On Time**, **On Exception**, or both.
3. Choose how you want to restore this tag's value from the persistence save file to the real-time database at application startup by clicking **Set Change Status ON** or **Set Change Status OFF**.
4. Click **OK**.

Persistence for a Domain

Domain persistence means that all persistent tags in a domain are saved the same way and restored the same way. This is in contrast to the individual method just described where each tag can be marked differently for saving and restoring. Configure persistence for a domain using both the Tag Editor and the Domain List.

> **Note:** The options selected in the Persistence and Change Bits fields apply *only* to those tags that have **Use Domain Settings** selected in their tag definition. These tags follow the domain configuration in the Domain List.

To configure Persistence for a domain, follow these steps:

1. Right-click your application and click **View > View Domain List**.
2. In the row containing the domain to be made persistent, click the **Persistence** arrow and select the method to save the tags' values:

None	The tags are not persistent.
Timed	Saves the values of the tags on a timed trigger.
Except	Saves the values of the tags whenever their values change.
Both	Saves the values of the tags both on a timed trigger and whenever their values change.

3. For the same domain, click the **Change Bits** arrow and select how to set the tags' change-status bits when their values are restored to the real-time database:

ON	Restores the tags with their change-status bits set to 1 (ON) after a warm start.
OFF	Restores the tags with their change-status bits set to 0 (OFF) after a warm start.

4 For the tags you want to mark as persistent, open the Tag Editor for that tag and select **Use Domain Settings** in the Persistence section. The tags will be saved in the Persistence save file and restored to the real-time database per the selections in the Domain List.

Note that if no tags are marked as Use Domain Settings, then the selections in the Domain List are ignored.

Configuring the Persistence Task

You must also configure the operation of the Persistence task itself along with marking Persistence for individual tags or domains.

Accessing

In your server application, open **System > Persistence > Persistence Save Information**.

Field Descriptions

Timed Save Trigger Tag used to trigger a save of the values of all tags marked as persistent by time.

When the tag is triggered at run time, the Persistence task reads all tags in the current domain instance configured to be saved on a timed basis and writes their values to the Persistence save file.

Leave this field blank only if no timed saves are required.

Note: If this field is left blank, you MUST fill in the Cache Buffers field. Not entering the appropriate information in a Persistence Save Information table will result in problems when creating the .CT file for Persistence. An error message will appear at run time, and the Persistence task will not run.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

Cache Buffers Indicates the number of buffers to set aside for the Persistence task's internal disk cache. The greater the number of buffers, the less the task writes to the disk, which improves performance.

Use the following guidelines to aid in determining the number of buffers:

- How often the data is changing
- How much of the data is changing
- The size of the data

While there is no definitive way to determine the ideal number of buffers, you can use the total number of tags of each data type that are persistent in the application to calculate an estimate of the maximum buffer size.

max buffer size = [(# of analog + digital)*2] + [(# of longana)*4] + [(# of float)*8] + [(# of msg)*len]

where **len** is the message length from the Persistence table. Then select a number of buffers multiplied by buffer size that equals this value.

Valid Entry: 0to 32766
0 = no disk caching is done
Default: 16

Buffer Size

Indicates the size, in bytes, of each buffer in the cache. The larger the buffer, the less the task writes to the disk, which improves performance.

Valid Entry: 64 to 32766
Default: 512

Message Copy Size

Indicates the maximum length allowed for message tags during persistent saves. FactoryLink uses this number when it reads tags from the real-time database and restores values during a warm start.

Valid Entry: 80 to 32766
Default: 2048

Backup Trigger

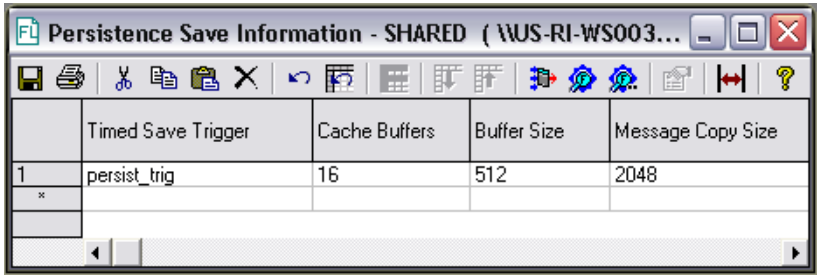
Name of a tag used to trigger a backup of the current persistent save file.

If the tag specified in this field is undefined, the Tag Editor appears when you click **Enter**.

At run time, when the application triggers the tag defined here, the Persistence task copies the current Persistence save file to a backup file.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

The completed table resembles the following sample.



In this example, when the value of **persist_trig** changes to 1 (ON), it triggers the Persistence task to save the values of all tags in the application configured as persistent by time. The number of buffers set aside for the internal cache is 16 with 512 bytes per buffer. A disk cache is a way to compensate for the slowness of the disk drive in comparison to RAM (memory). The Persistence task's cache process speeds up computer operations by keeping data in memory. Rather than writing each piece of data to be saved to the hard disk, the task writes the data to its internal disk cache (reserved memory area). When the cache process has time, it writes the saved data to the hard disk.

The maximum length for message tags during persistent saves is 2048 bytes. When **persist_backup** is triggered, Persistence copies the current Persistence save file to a backup file.

PERSISTENCE TASK START ORDER

This section describes some possible effects of warm restarts on the application depending on when Persistence starts and what has happened to the other tasks during the startup process.

Caution: This discussion applies only if the TASKSTART_? tags from the System Configuration table are made persistent.

Example 1: Persistence starts last and shuts down first

After the application starts, the values of the TASKSTART_? tags are 1, so Persistence saves a 1 as their last known value. At shutdown, because Persistence stops first, Persistence does not see the change in value of the TASKSTART_? tags from 1 to 0 (zero), so the saved values remain as 1. On a warm start of the application, the TASKSTART_? tags for all tasks running at shutdown are restored to 1 and therefore, their tasks will start. It is important to note that these same tasks will be started regardless of their "R" flag settings in the SYS.CT file. and that there are no manual starts or terminations.

Example 2: Persistence starts first and shuts down last

Because Persistence starts first, it sees the application starting and, therefore, sees the values of the TASKSTART_? tags at 0. Because Persistence stops last, it saves a 0 as the last known value of the TASKSTART_? tags if a termination happens during the startup process. On a warm start of the application, none of the tasks start because all of the TASKSTART_? tags have a last known value of 0.

The shutdown order is more significant than the startup order if the tags are saved on change. In general, specify the Persistence task to shutdown first (and therefore, start last) so the saved values in the Persistence save file reflect the last known running state of the application at shutdown. Then, the warm start restores it to that state, which is the purpose of the Persistence task.

PERSISTENCE AND DIGITAL TAGS

The way a digital tag is used in an application affects how that tag is configured for Persistence. Digital tags are often used to trigger some action in an application. Examples include starting a Math & Logic procedure or starting a FactoryLink task. When a digital tag is triggered (its value is changed from 0 to 1 or force-written to 1), FactoryLink starts the associated procedure or task. When FactoryLink is warm-started, these tags are restored to their last saved value. Configuring a digital tag as persistent with its value to be restored with Change Status ON can be used to start any procedure or task associated with this tag after the system is initialized.

However, the digital tags RTMCMD and RTMCMD_U, cannot be made persistent because, when the value of these tags is set to 1, the FactoryLink system shuts down, which makes these tags persistent and immediately shuts down the system.

Note that the R (Run) flag for each task in the System Configuration Information table supersedes the value of the digital start trigger associated with a task.

The following examples show the relationship between the R flag in the System Configuration Information table and the restored value of a digital tag.

Example 1

The R flag is NOT set for task A, and the digital start trigger associated with task A is defined as persistent by Exception (always updated) with Force Change Status ON if:

- Task A is running when the system is shut down, then the value of the task's digital start trigger is 1. When a warm start is performed, the system restarts task A because the value of the digital start trigger is restored to 1.
- Task A is not running when the system is shut down, then the value of the task's digital start trigger is 0. When a warm start is performed, the system does not restart task A because the value of the digital start trigger is restored to 0.

Example 2

The R flag IS set for task A and the digital start trigger associated with task A is defined to be persistent by Exception (always updated) with Force Change Status ON if:

- Task A is running when the system is shut down, then the value of the task's digital start trigger is 1. When a warm start is performed, the system restarts task A because the task's Run flag is set.
- Task A is not running when the system is shut down, then the value of the task's digital start trigger is 0. When a warm start is performed, the system still restarts task A because, even though the value of the digital start trigger is restored to 0, the task's Run flag is set and the Run flag supersedes the restored value of the digital start trigger.

PERSISTENCE SAVE FILE NAME

The persistent data is saved in a unique Persistence save file for each domain instance. The Persistence save files have the extension .PRS and are located in the **/FLAPP/FLNAME/FLDOMAIN** directory.

where

FLAPP	Translated application environment variable.
FLNAME	Translated application environment variable.
FLDOMAIN	Translated domain environment variable.

The name of each Persistence save file is {FLUSER}.PRS where FLUSER is the translated environment variable for the domain user name. The Persistence save file contains the saved values for that domain user.

For example, in Windows, where the FLRUN.BAT file sets the Shared FLUSER environment variable to SHAREUSR, but the User domain FLUSER environment variable remains at the default setup in the AUTOEXEC.BAT file, the Shared persist file is named SHAREUSR.PRS and the User persist file is named FLUSER1.PRS.

The Persistence backup files are in the same place and have the same name, except they have the extension .BAK.

EDITING TAG PERSISTENCE SETTINGS USING BH_SQL UTILITY

It may be useful for users to be able to make mass edits to the current Persistence settings for defined tags in the OBJECT configuration table, such as changing the field entries for all tags that currently are blank to a specific setting, such as NONE. This can be done using the BH_SQL utility provided with all FactoryLink systems. This utility has the following functions:

1. Modifies the OBJECT.CDB file in the **FLAPP** directory
2. Modifies the **TAGPERWHEN** field in that file

TAGPERWHEN (meaning Tag is saved when) is the text equivalent to the buttons on the Tag Editor when defining a tag or using **CTRL+T** to view the tag definition. The possible values are:

- NONE—tag is not persistent
- Left blank—same as NONE
- DOMAIN—save based on domain Persistence definition as configured in the Domain configuration table.

- TIMED—save on timed trigger
- EXCEPT—save on change
- BOTH—save on timed trigger or change

The procedure updates the table changing all instances of a specific entry in the **TAGPERWHEN** field at one time to a new value.

Prior to executing the instructions below, we recommend you make a backup of the application using the FLSAVE utility or some other backup utility. At least make a backup copy of the OBJECT.CDB and OBJECT.MDX files so if anything goes wrong during the procedure, the backup can be restored with no damage done to the application. The general syntax can be modified to update the Persistence setting for any group of tags from the current settings to any valid new setting as a group, by varying the literal values in the first and second instance of **tagperwhen = '????'**.

Perform the following steps to use the BH_SQL utility:

1. Type the program name at a prompt for all systems except MS Windows. For MS Windows, run the program from **Start > Run**.
2. At the BH_SQL prompt, type **SQL > connect flapp** and press Enter.

   "flapp" is the actual path to the FLAPP directory as defined in the environment variable.
3. Type **SQL > update object set tagperwhen = 'NONE' where tagperwhen = ''** and press Enter.

   Quote marks are all single quotes not double quotes. The first instance of **tagperwhen = 'NONE'** is the desired new value for the field and the second instance is the current value of the field (in this case a blank entry). This command finds all records in the OBJECT table for which the current tag Persistence setting is blank and changes all the settings to NONE.

   Use the following command if you have a large number of tags configured to be saved as defined for the domain configuration and you want to change the setting for all of these tags to be saved individually when they change value or on exception.

   SQL > update object set tagperwhen = 'DOMAIN' where tagperwhen = 'EXCEPT'
4. After all desired changes are made, type **QUIT**.

ERROR MESSAGES

Error Message	Cause and Action
Can't get a task ID for %s	Cause: Could not connect to a running FactoryLink application. Action: Ensure the application is running, then try again.
Can't open the file or the /FLAPP/CT/PERSISTENCE.CT file does not exist or is corrupt	Action: If the file exists, delete it and restart the application to recreate the file. If the file does not exist, create it by restarting the application.
Can't use backup save file	Cause: If the RESOLVE program cannot use the Persistence save file (*.PRS) because it is corrupted, the program looks for the Persistence backup file (*.BAK). If RESOLVE cannot use the .BAK file, it generates this message. Action: Delete the original save and backup files and the task creates new save file and a new backup file.
Error deleting the file	Cause: The specified file cannot be deleted. Another developer may be using it or the file is read-only. Action: Check to see if the file is being opened by another developer or if the file is read-only.
Error getting FLINK environment variables	Cause: The variables FLDOMAIN, FLNAME, and FLUSER are not defined. Action: Define the variables from the system prompt using **set (flvar)=(flvardef)** **flvar** is the FactoryLink environment variable FLDOMAIN, FLNAME, or FLUSER. **flvardef** is the developer-defined name for this variable.
Error reading CT file: *filename*	Cause: Either the file **/CT/PERSISTENCE.CT** is corrupt, or the .CT script file (**/FLINK/CTG/PERSISTENCE.CTG**) and the FactoryLink Run-Time version are not the same version. Action: Delete the file **/FLAPP/CT/PERSISTENCE.CT**. Restart the application to rebuild the file.
Error reading Persistence file: *filename*	Cause: Either the file is corrupt or the disk is damaged. Action: Inspect the file. Run CHKDSK or any disk diagnostic program to locate problems with the disk drive.

Error Message	Cause and Action
Error reading RTDB: %d	Cause: You tried to read a tag value from the database and the kernel returned this message with an error number. Action: Shut down and restart the application and try to read the value again.
Error seeking in file: *filename*	Cause: Either a hardware error or a corrupted system exists. Action: Contact your technical support representative.
Error writing to file: *filename*	Cause: Either a hardware error or a corrupted system exists. Action: Contact your technical support representative.
Invalid CT file structure in: *filename*	Cause: The specified .CT file is corrupted. Action: Run **ctgen -r** to rebuild the .CT files and try again.
Invalid or corrupt save file	Cause: The Persistence save file (*.PRS) is corrupt. The task looks for and attempt to use the Persistence backup file (*.BAK). Action: No action required - information message only.
Out of RAM	Cause: Not enough RAM available to perform this task. Action: Close any unnecessary windows or programs, such as the Client Builder or any text editor. Add RAM to the system if this error message occurs often.
%s is not a FactoryLink Persistence file	Cause: The specified file is not in the Persistence save file format and does not have a .PRS extension. Action: Remove the offending file. If the Persistence backup file (*.BAK) exists, copy it to the save file with a .PRS extension.
Save file was not closed properly	Cause: The task generates this message when it begins to copy the Persistence backup (*.BAK) file on the Persistence save (*.PRS) file. Action: No action required - information message only.
Task Initialization failed	Cause: The Persistence task is already running, a key is not installed, the wrong key is installed, or you are not authorized to have the key. Action: Check that the proper key is installed if you are authorized to have the key.

Chapter 16 . . . .
PowerNet

PowerNet allows you to share Real-time Database tags among FactoryLink applications running on the same or different workstations or nodes.

One FactoryLink application can act as a client and/or server. This application can serve other FactoryLink applications by providing needed information. As a client, the application can use information provided by other FactoryLink applications.

On platforms where FactoryLink supports multiple applications running on the same computer, you can run multiple instances of PowerNet. See the *Fundamentals Guide* for a discussion about the multiuser architecture.

The configuration tables used to configure the PowerNet task are completed in the Shared domain. Currently, the only network protocol supported by PowerNet is TCP/IP.

> **Note:** PowerNet was an early FactoryLink task for sharing data between nodes on a network. In a later version of FactoryLink, the Virtual Real-Time Network and Redundancy (VRN/VRR) task was introduced. VRN/VRR has all of the functionality of PowerNet and is more flexible. PowerNet is still supported, but if you are starting a new application, it is recommended that you use VRN/VRR instead. For more information, see “Virtual Real-Time Network and Redundancy” on page 521.

Operating Principles

This section describes what initiates data transfer between a server application and a client application.

Startup

As each client attaches to a server application, all of the data shared between the server and client applications is transmitted from the server to the client. This ensures the client contains up-to-date data immediately upon starting up. This also occurs at reconnection in the event a connection is lost between the client and server.

Server-to-Client Data Transfer

Data transfer from the server to the client is configured by one of the following two methods:

- Exception Data—Transmits data to the client only when data has changed in the server application.
- Polled Data—Transmits data to the client on a fixed interval, a dynamic interval, or at any event the client application generates.

Client-to-Server Data Transfer

A data connection to an external domain may be configured as read only (the default) or read/write. Read/write connections allow data that has changed in the client application to be written back to the server application.

Domains that include mailbox tags should be READONLY, not READWRITE. If a domain containing a mailbox tag is READWRITE, PowerNet will ignore the mailbox tag in the write-back connection. The task will still run, but PowerNet will print a warning message

Tag Naming Conventions

Tag names referencing data tags from other applications use the following format.

exdomain:`tagname`{[sub1] {,[sub2],...}}

Tag Type Conversion

Tags that reference data in a remote application can be of a different data type from the tag definition in the remote application.

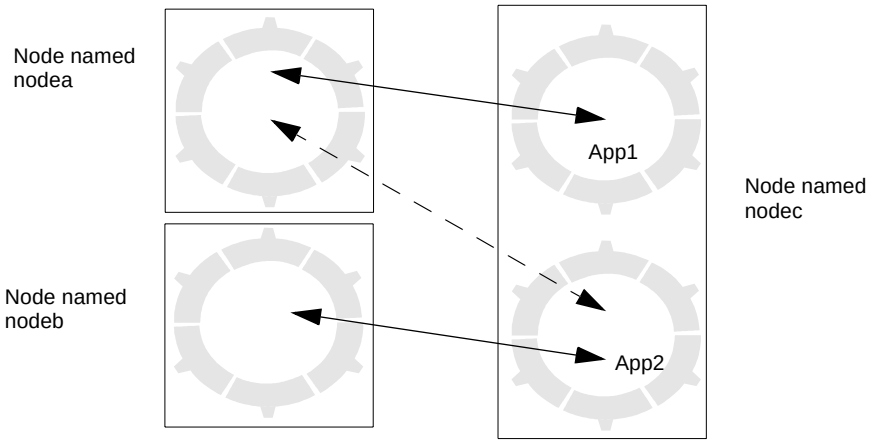
NETWORK SOFTWARE

Perform the following steps to configure network software:

1 Design your network topology. Include the following information for each node (or a TCP/IP host).

- Node name
- IP address
- Client/server connections

For example, the following drawing shows a network with three nodes: nodea, nodeb and nodec.



In this example, a single FactoryLink application is running on nodea and nodeb.

- Two separate FactoryLink applications are running on nodec.
- The first instance of FactoryLink running on nodec references data on nodea.
- The second instance of FactoryLink running on nodec references data on nodeb.

2 Add the names of all nodes in the network that share FactoryLink data in the TCP/IP hosts file. Do this for each client and server node running FactoryLink.

The format of host file entries is

network_address *nodename ALIAS*

where

network_address — The network IP address for the node.

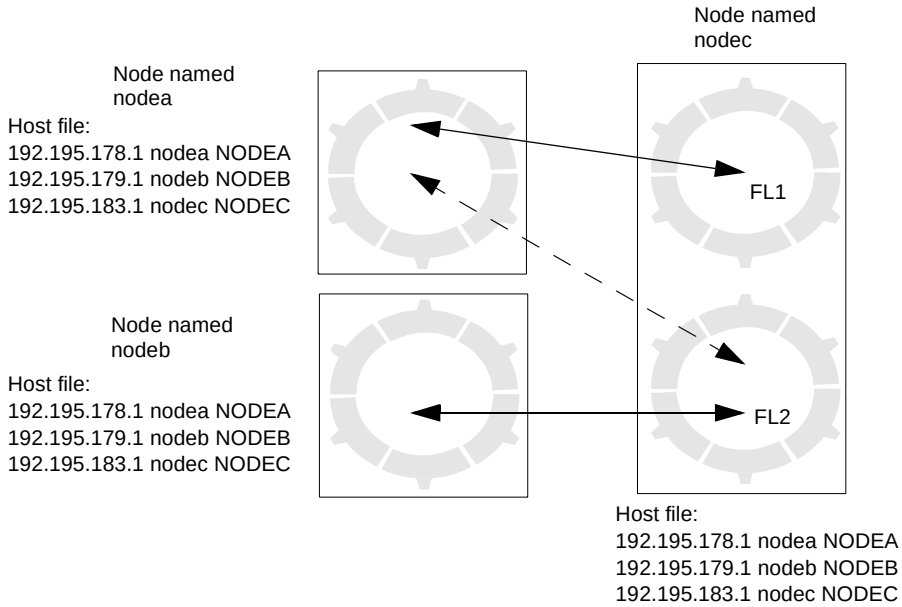
nodename — The lowercase specification of the name assigned to the node and can be up to 256 characters.

ALIAS — The uppercase specification of the name assigned to the node.

For example, the following host file identifies the nodes in the example.

```
192.195.178.1 nodea NODEA
192.195.179.1 nodeb NODEB
192.195.183.1 nodec NODEC
```

Two server nodes are named nodea and nodeb and one client node named nodec.



3 Define the environment variable FLHOST for your operating system that corresponds to the local host name. This environment variable must be set for each application instance.

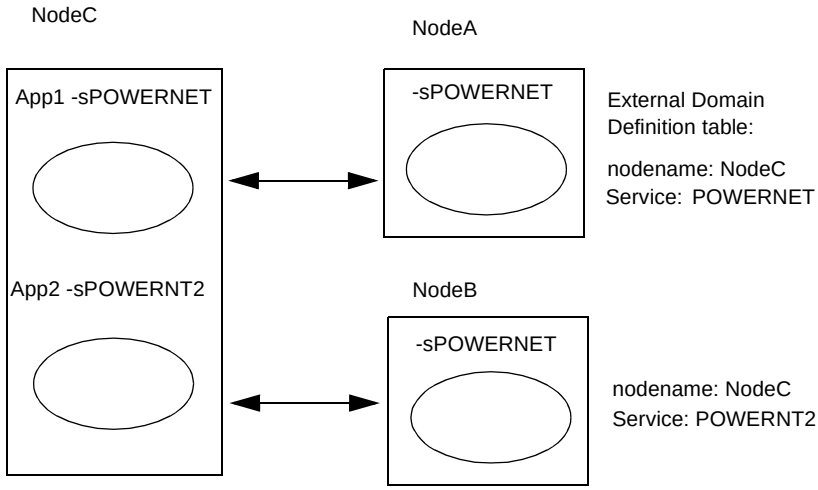
Alternatively, the application can be passed a program argument at run time to define the local host name. You define this argument in the System Configuration Information table, discussed on page 399.

4 Add the name(s) assigned to each PowerNet service running on the node in the TCP/IP services file. Do this for each client and server node running FactoryLink. Refer to the appropriate vendor documentation for more information on configuring services.

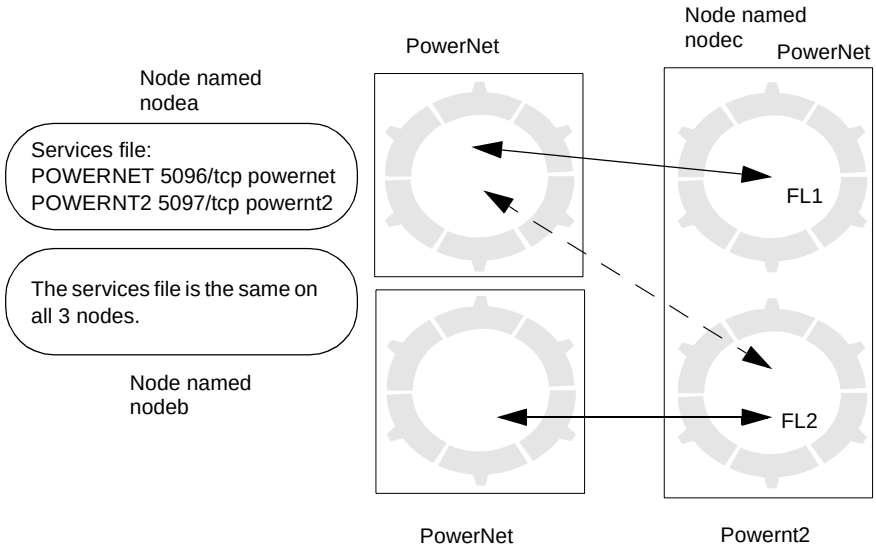
One services file is associated with each node. The services file should contain the names of PowerNet services for each FactoryLink application running on the node. The default service name is POWERNET, which can be used if only one application on a node is running PowerNet.

Each instance of PowerNet must use a unique service name if more than one application running PowerNet exists on a node. The service name the local PowerNet uses, specified using the `-s` command line parameter, must match the service name (*Remote Service Name or TAG) specified in the External Domain Definition table for that domain on the remote node.

The following graphic illustrates this.



For the best results, the services files should be identical on all nodes.



The format of services file entries is

SERVICE port_num alias

where

SERVICE Is the uppercase specification of the name assigned to the service running on the node. This name can be from 1 to 8 characters and must be unique for each service defined for a single node. The default is POWERNET.

port_num Is a unique number assigned to reference the port number to TCP/IP. This number must be unique for each service defined for a single node. The recommended port number is 5096; however, any number can be used as long as it is consistent across all services files.

alias Is the lowercase specification of the name assigned to the service running on the node.

For example, the following services file identifies the services for the nodes in the previous diagram:

```
POWERNET 5096/tcp powernet
POWERNT2 5097/tcp powernt2
```

EXTERNAL DOMAIN DEFINITION TABLE

PowerNet requires only a mapping of the external domain name to actual network node and server names and to be running on both the client and server applications to access an external domain. This is configured through the External Domain Definition table. Additionally, this table is used to configure update type, update rate, and data transfer.

Accessing

In your server application, open **Networking > External Domain Definition > External Domain Definition**.

Field Definitions

Domain Name

Name of the external domain. Add an entry for each unique external domain. Use this name as part of the remote TAG reference. For example: NODEB:temp. This logical representation of a FactoryLink application is limited to 256 characters. For example, a client application can access information from two logical server applications, NODEB and NODEC.

Valid Entry: domain name

***Network Node Name or TAG**

Node name where the FactoryLink server application represented by this connection resides. This name can be a tag name or a constant preceded by a single quote.

If you enter a constant, make the constant a valid TCP/IP host name. If this field references the local node, PowerNet performs local tag value transfers.

Valid Entry: tag name or constant
Valid Data Type: message

***Remote Service Name or TAG**

Name of the service the server application represented by this connection uses. This name can be a tag name or a constant preceded by a single quote.

If you enter a constant, the name must match the name assigned to the instance of PowerNet used by the server application as defined in the TCP/IP services file.

If you leave this field blank, the services name defaults to POWERNET.

Use the *-sservice_name* program parameter to specify the service name for PowerNet in the local application.

Valid Entry: tag name or constant
Valid Data Type: message

Update Type
Method used to send data from the server application to the client application. This can be one of the following:

- **EXCEPTION** Transfers data when tag change status bits are set. This is the default.
- **POLLED** Transfers data according to the interval defined in the **Update Rate** field. Only values that have changed since the last poll are transferred. You must specify an update rate in the next field if you specify polled.

***Update Rate or TAG**
Number of seconds between updates if the update type is polled. This can either be a constant or a tag name.

If you specify a digital tag, the tag acts as a poll trigger. A transition from 0 to 1 or a forced value of 1 initiates a poll.

If you specify an analog tag, the value of the tag specifies the number of seconds between polls at run time. This may be changed at run time.

If you leave this field blank, the default is 0 with polling disabled.

Valid Entry: tag name or constant
Valid Data Type: digital, analog

Data Transfer
Read/write privileges between the client and server.

- **READONLY** The client application can only receive data from the server application. If configuring the Distributed Alarm Logger task to use with PowerNet, you must use READONLY as the data transfer type.
- **READWRITE** The client application can receive data from the server application and write data back to the server if the value of the tag changes in the client application.

Status Message TAG
Tag updated by PowerNet that contains the status of the connection for the client is stored in. Possible status messages are:

Create Client—PowerNet has created the client object and is waiting to connect

Deleting Client—PowerNet is deleting the client object. This means no tags are associated with this domain.

Connecting—The client object is in the process of connecting.

Binding—The client object has connected and is in the process of binding to the remote PowerNet.

Abort Connection—The client has aborted the connection. This usually means the server is not available.

Connect Timeout—The server did not connect in the time allowed.

Connected—The server has connected and bound. This is the normal state for PowerNet.

Bind Reply Timeout—The client did not receive a message from the server after the connect in the time allowed.

Receive Reply Timeout—The server has not sent any response in the time allowed. The server has most likely shut down abnormally.

Send Failed—A problem has occurred sending the data.

Receive Disconnect—The server has sent a disconnect message to the client. The server will send this message if PowerNet is shut down on the server's host machine.

Received Error—An error occurred during a read.

Valid Entry: tag name
Valid Data Type: message

Force Writes Allows a client to send a forced write of a digital tag to the server to trigger an event.

SYSTEM CONFIGURATION INFORMATION TABLE

Run-time parameters that control how PowerNet starts and executes at run time are defined in the System Configuration Information table. Each FactoryLink application must have PowerNet running, even if two applications reside on the same node.

Accessing

In your server application, open **System > System Configuration > System Configuration Information**.

Copy and paste the last row of the System Configuration Information table into the empty row just below it if a row for PowerNet does not exist.

Field Definitions

Flags **FR** to instruct the task to start automatically at run time.

Task Name **POWERNET** to identify the task to the system.

Description Change the description to reference PowerNet.

Start Trigger... Display Description Increment the array offset by 1 for all entries ranging from **Start Trigger** to **Display Description**.

Start Order	One (1) to ensure the task starts up appropriately at run time.
Executable File	**bin/powernet** to specify the location of the executable file.
Program Argument	Any desired program arguments to control how the task functions at run time.

PROGRAM ARGUMENTS

All of the options are optional, but if the -h is not used to set the host name, the FLHOST environment variable must be used instead to specify the local host name. Arguments are case sensitive.

Argument	Description
-b<#>	Set transfer buffer size. (# = kilobytes) Where # is the buffer size (Default = 512) The buffer size is the maximum packet size PowerNet sends across the network. PowerNet only sends the data that it has to up to the buffer size. If all the data cannot fit in one packet, PowerNet breaks the data into several packets. If you increase the buffer size on the Windows platform, remember to increase the TCP/IP buffer size also.
-c<#>	Set connect time. (# = seconds) Where # is the amount of time in seconds for PowerNet to try to reconnect (Default = 10) The -c option is the number of seconds between connect tries. PowerNet continuously tries to connect to the server. This option could be useful if a server will not be running for a long period of time and you do not want PowerNet to continue trying to connect.
-d<#>	Set debug verbose level. (# = 0 to 4)
-h <H>	Set host name. (H = Host name) The local host name may be specified either with the -h parameter or in the environment variable FLHOST. It must be specified in one of the two places, or PowerNet will not run. If both are specified, the command line (-h) overrides the FLHOST variable.
-i<#>	Set timeout for bind wait. (Time to wait for network connection.) (# = number of milliseconds to sleep after binding every 20 tags) This parameter is important in applications where PowerNet is binding a large amount of tags and is using too much CPU time. This parameter slows the process down but allows other tasks to function normally during PowerNet binding.

Argument	Description
-l	Enables logging of debug information to a log file.
-m<#>	Set timeout for message transfer. (# = seconds) Where # is the amount of time in seconds for a data transfer default: 1 The -m option is the amount of time allowed for a transfer of data to occur. When PowerNet is used with a modem, this option can be set to allow for the data to be completely transferred.
-n<#>	Set number of sessions. (# = max number) Where # is the number of sessions default: 32 The number of sessions specifies the maximum number of connections PowerNet may make. For each read-only domain specified in the External Domain table, PowerNet makes one connection. For each read-write domain, PowerNet makes two connections. Also allow sessions for incoming connections where other PowerNet clients are connecting to the local PowerNet as a server.
-p<#>	Set timeout for Send. (# = seconds) Where # is the amount of time in seconds for a send to occur. (Default: = 15) PowerNet could possibly be in exception mode and has no need to send a message to the other machine. To prevent the connection from being disconnected because of no new messages, PowerNet sends an alive message. The -pn option can specify how often this message is sent. (Note: Do not set the send time greater than the receive time. This situation can cause many disconnects to occur.)
-r<#>	Set timeout for Reply. (# = seconds) Where # is the amount of time in seconds for a connect to occur (Default = 30) The -r option is the amount of time allowed by the client for a response from the server after a connection. The option is very useful if larger numbers of tags are being sent from the server to the client. The server must do all the initialization for each tag before it responds to the client. Setting this option will allow the server to accomplish the initialization before the time-out period.

Argument	Description
-s <S>	Set service name. (S = Service name) The service defaults to POWERNET. The service name is the name in the TCP/IP services file that tells PowerNet which TCP/IP port to listen on. If more than one PowerNet is running on a machine, each must have a different service name.
-t<#>	Set timeout for Receive. (# = seconds) Where # is the amount of time in seconds for a receive time out (Default = 30) The -t option allows the user to modify the time that PowerNet expects for either data or an alive message from a connection. If either data or an alive message has not been received, PowerNet assumes the connection is lost and aborts the connection.
-v	Insert timestamp at beginning of each debug statement.
-w<#>	Wraps log file every # messages.
-y<#>	Closes and reopens log file every # messages.

TROUBLESHOOTING

The PowerNet task can display and log information during run time. Customer Support uses this information to determine and resolve the user's problems. The amount and the content of the information being logged is controlled by the command line options. PowerNet was implemented in three layers, PowerNet, NSI class, and NSI. NSI stands for network services interface and is the TCP/IP specific layer. The NSI class layer is an intermediate layer between NSI and PowerNet. Each layer has its own topics and levels (NSI class does not have topics). If you have a PowerNet problem and are working with Customer Support, they will tell you which categories and levels to use to produce the most helpful log file.

The following are examples of the command line debug options for PowerNet:

-B***n***	Display the messages related to the topic B up to level ***n***
-C***n***	Display the messages related to the topic C up to level ***n***
-BN***n***	Display the NSI messages related to the topic B up to level ***n***
-CN***n***	Display the NSI messages related to the topic C up to level ***n***
-d***n***	Display the messages from all topics up to the level ***n***
-o***n***	Display the messages from NSI class up to the level ***n***
-x***n***	Display the messages from NSI up to the level ***n***
-l	Log the displayed messages to the file
-v	Insert a timestamp in the beginning of each message
-w***m***	Wrap the log file every ***m*** messages (see more detailed descriptions below)
-y***p***	Perform closing and reopening of the log file once per ***p*** messages

Note: The options are case-sensitive, **-D3** is not the same as **-d3**. The use of **-dx** supports the old style logging messages where all categories are displayed at level **x**.

The following is a description of the topics and levels that are currently in use:

B - Binding

1 - errors
2 - warnings
3 - bind request/response was sent/received from NODE
4 - bind logic
5 - contents of bind request/response
6 - bind logic (more detail)

C - Connection/Disconnection

1 - errors
2 - warnings
3 - connected or any reason for disconnecting
4 - state of session, how many nodes are on-line
5 - received/sent connect packets
6 - contents of connect packet, more details

D - Data/tags

1 - errors
2 - warnings
3 - type and count tags in packet
4 - value of tag
5 - data conversion
6 - more details of data conversion

F - FactoryLink activities and triggers

1 - errors
2 - warnings
3 - setting FL status and message, shutdown
4 - waiting on triggers/which trigger went off
5 - more details of waiting on triggers/which trigger went off
6 - updating monitor tags

I - Initialization (CT files, LOCAL file, GROUPS file)

1 - errors
2 - warnings
3 - LOCAL file parameters, my system type and version
4 - dumping CT and GROUP information
5 - mode details of dumping CT and GROUP information
6 - loading groups
7 - Maxlength, bufsize, e.t.c. after netinit
8 - entering/exiting initialization procedures

R - Receiving

1 - errors
2 - warnings
3 - a packet is received from NODE
4 - packet header
5 - processing the received packet, calling receive

S - Sending

1 - errors
2 - warnings
3 - a packet is send to NODE
4 - packet header
5 - checking for time-outs, waiting for pkts
6 - send logic
7 - send logic
8 - more details
9 - detailed information about room left in the packet
10 - mailboxes

M - Miscellaneous

1 - errors
2 - warnings

CN - Connection/Disconnection in NSI layer

1 - errors
2 - warnings
3 - connecting or disconnecting events
4 - more details
5 - even more details

IN -Initialization in NSI layer

1 - errors
2 - warnings
3 - Initialization events
4 - more details

NN - Network activities in NSI layer

1 - errors

2 - warnings

3 - session state

4 - session information

5 - more details

SN - Sending in NSI layer

1 - errors

2 - warnings

3 - sending events

4 - more details

RN -Receiving in NSI layer

1 - errors

2 - warnings

3 - receiving events

4 - more details

In addition to the topics and levels, the messages comply to a certain format:

- All error messages begin with the word ERROR.
- All warning messages begin with the word WARNING.
- Information and debug messages do not have a specific format.

-w***m*** Wrap the log file every ***m*** messages

When this command line argument is specified along with **-l** argument, the logging mechanism keeps the size of the log file under **m** messages. The **<name>.log** file always contains no more than **m** most recent messages, when the (m + 1) message comes, the **<name>.log** file gets renamed to **<name>.111**, and the new **<name>.log** file gets created. In addition, the very first **m** messages are stored forever in file **<name>.000**. So, in common cases, three files are always on a disk: **<name>.000**, **<name>.111**, and **<name>.log**.

The default is to let the log file grow indefinitely with the extension .log except on the Windows platform where the default is a maximum number of messages equal to 65535, and the maximum number of messages may not be set higher than 65535. On other platforms, the maximum number of messages may be set higher than 65535.

The **-w** option is particularly useful when tracking a PowerNet problem that takes a long time to reproduce. This option prevents the log file from consuming all available disk space.

Example

-hFLHP2 -b1024

Local host name: FLHP2
Buffer size 1024

C:\> fllanrcv -C3 -R1 -D4 -l -s2 -f -w400

Will be interpreted as following:

- Run FLLAN Receive task displaying and logging:
  - error messages, warning messages and major info messages related to connecting/disconnecting activities
  - error messages related to receiving
  - error messages, warning messages, major and minor info messages related to Tags and Data
- Include File and Line in each message.
- When logging every second message, close and reopen log file.
- Wrap the log file every 400 lines.

ERROR MESSAGES

Error Message	Cause and Action
Cannot initialize NSI interface!	Cause: Network initialization failed Action: Verify the TCP/IP network is installed and running. Verify the POWERNET service name is added to the services file and the local and remote node names are known to the network either through the hosts file or the Domain Name Server.
Can't open CT file extrndom.ct	Cause: PowerNet cannot find the EXTRNDOM.CT file because the External Domain Definition table is not configured. Action: Configure the External Domain Definition table.
Error creating service object! NSI error %4x **Error exposing service object! NSI error %4x**	The network has a problem. Any of the following conditions can cause this error: Cause: No more sockets are available. Action: Shut down other network processes to free up sockets or increase the number of available sockets if possible. Cause: The POWERNET service name cannot be found in the services file. Action: Check the services files for the POWERNET service name or other service name being used. Cause: The remote node name cannot be found in the hosts file. Action: Check for the correct remote node name and address in the hosts file or Domain Name Server. Ensure the node name is in both lower and upper case (one being the alias). Cause: Another process is already bound to the same service name. Action: Check if another copy of PowerNet is already running or another process could be using the same service name. If multiple copies of PowerNet are run on a single node, they must all use different service names.

Error Message	Cause and Action
Error reading CT header **Error reading CT index** **Error reading CT record**	Cause: The EXTRNDOM.CT file is corrupt. Action: Rebuild the file by deleting the EXTRNDOM.CT file and restarting FactoryLink.
Mailbox tag tag_name is ignored in the write-back domain	Cause: A domain configured as READWRITE contains at least one mailbox tag. The message is displayed to warn the user that mailbox tags are ignored in the write-back domain. All other tags will still work. Action: None is needed, but to get rid of the warning message, change any domains containing mailbox tags to READONLY. To change the domain to be READWRITE, remove any mailbox tags from that domain and create a new READWRITE domain for the mailbox tags.
No local host or FLHOST variable supplied	Cause: The FLHOST variable is not supplied. Action: Either set the FLHOST variable before running PowerNet or pass it to PowerNet using the **-s** argument.
Out of RAM	Cause: The process could not allocate needed memory. Action: Close any unnecessary windows or programs, such as the Client Builder. If this error message occurs often, either add more memory to the system or configure the existing memory differently.
Unsupported data type for PowerNet	Cause: A tag of an unsupported type is configured. Action: Contact your technical support representative.

Chapter 17 . . . .
PowerSQL

The PowerSQL (Structured Query Language) task works in conjunction with the historian task to allow an application to access data in an external relational database through a result window. PowerSQL offers the following features:

- Allows data in an external relational database to be manipulated from within FactoryLink
- Allows an application to send and retrieve data to and from external database tables, including those created outside FactoryLink
- Allows you to define tags referenced by PowerSQL in arrays as well as individually
- Allows you to execute SQL statements generated in Math & Logic
- Allows you to execute database-stored procedures for database servers that support them
- Allows you to processes SQL statements that are entered in a FactoryLink message tag

Operating Principles

PowerSQL is a historian-client task that communicates with historian through mailbox tags to send and receive historical information stored in an external database using SQL.

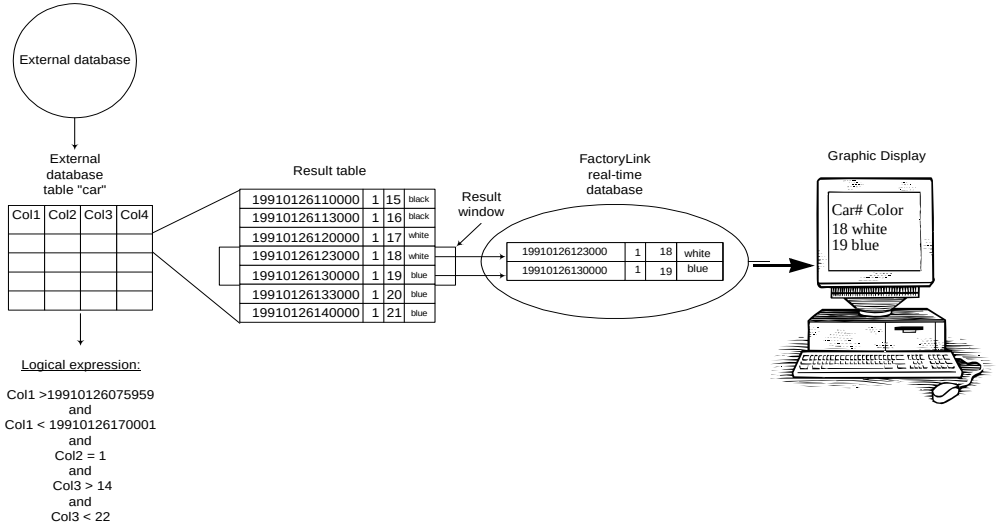
PowerSQL retrieves data in a relational database by generating an SQL SELECT from the data specified in a FactoryLink configuration table and placing it in a temporary table called a result table. The FactoryLink application can view and modify the retrieved data in the result table through a result window. A result window is a sliding window that maps data columns in a relational database table to FactoryLink tags. The result window views selected portions of the result table.

For example, if a graphic screen is used to display the result window, it can display as many rows of data from the result table as there are tags in the two-dimensional tag array. If there are more rows in the result table than in the result window, the operator can scroll through the result table and see each row of the table in the result window.

PowerSQL modifies data in a relational database by generating UPDATE, DELETE, and INSERT SQL statements from the data specified in a FactoryLink configuration table. The PowerSQL task also executes SQL statements generated by the user in a FactoryLink message tag.

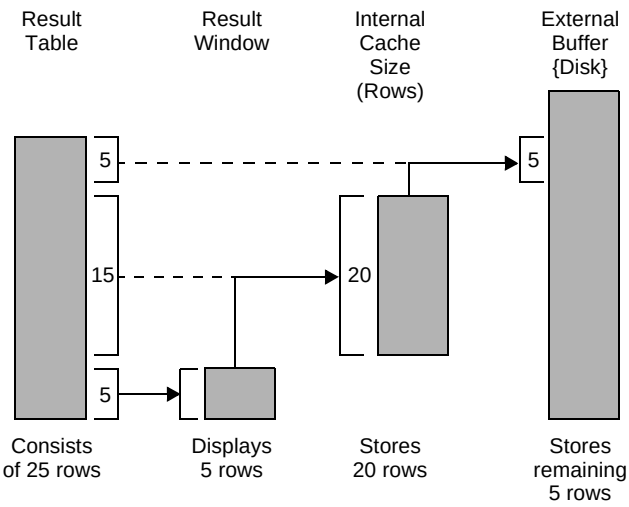
PowerSQL can read from and write to an entire array of tags in one operation. The relationships among the external database, the result table, the result window, the real-time database, and the graphic display are displayed in Figure 17-1.

Figure 17-1 PowerSQL Relationships



An internal buffer stores the rows of the result table in RAM. An external buffer stores the overflow of rows from the internal buffer on disk. This allows the operator to scroll back up through the result table. The buffers are shown in Figure 17-2.

Figure 17-2 Buffers Used in PowerSQL



In this example, as the operator scrolls through the result table, the rows of the result table flow into the internal buffer to be stored in memory. Because, in this case, the result table consists of 25 rows and the internal buffer can store only 20 rows, when the internal buffer is full, the excess rows in the internal buffer flow into the external buffer to be stored on disk.

LOGICAL EXPRESSIONS

You use logical expressions to specify the data in a relational database to view or modify. For the purposes of PowerSQL, a logical expression is a command containing a standard SQL WHERE clause. To make a logical expression flexible at run time, use the name of a message tag whose value is a WHERE clause. If viewing all data from a column in a relational database table, you do not need to specify a logical expression.

You must know how to write a standard SQL statement to configure PowerSQL. For information about writing SQL statements, refer to any quick reference SQL guide or the user manual for the relational database in use.

To select data from a database table, a logical expression works in conjunction with the table's column name and logical operators to form an SQL WHERE clause. The WHERE clause specifies which rows in a database table to place in the result table.

The following table represents part of a sample database table CAR.

TRANDATE	CONVEYOR	CARNUM	COLOR
20040126100000	1	13	silver
20040126103000	1	14	black
20040126110000	1	15	black
20040126113000	1	16	black
20040126120000	1	17	white
20040126123000	1	18	white
20040126130000	1	19	blue
20040126133000	1	20	blue
20040126140000	1	21	blue

A sample WHERE clause referencing the previous table CAR is

TRANDATE > '20040126075959' AND TRANDATE < '20040126170001' AND CONVEYOR = 1 AND CARNUM > 14 AND CARNUM <19

In this example, the WHERE clause requests:

What were the colors of cars 15 through 21 on conveyor 1 painted between 8:00 A.M. and 5:00 P.M. on January 26, 2004?

From this WHERE clause, the relational database places the following values in a result table.

19910126110000	1	15	black
19910126113000	1	16	black
19910126120000	1	17	white
19910126123000	1	18	white

If the view size of the result window is 2, the result window writes the values of the tags in two rows to the real-time database. When the data reaches the real-time database, other FactoryLink tasks can read it and write to it, and an operator can view the data on a graphics screen.

PowerSQL performs five types of operations:

Select Uses an SQL select statement to select and retrieve data from a relational database to be displayed in a result table.

Update Updates the data in the result table and external database. PowerSQL can perform two types of update operations:

Positional—Updates the current row (row at which the cursor is currently pointing) of data displayed in the result window.

Logical—Updates the data described by the logical expression.

Insert Inserts a row of data in a database table. There are two types of insert operations:

Auto Create flag—If this flag is true, an insert operation is executed when an update operation modifies no rows in the database table.

Insert trigger—Inserts a row of data when the trigger is set.

Delete Deletes a row from the result table and external database. PowerSQL can perform two types of delete operations:

Positional—Deletes the current row (row at which the cursor is currently pointing) of data displayed in the result window.

Logical—Deletes the data described by the logical expression.

Other SQL statements Using a message tag, you can make ad hoc queries.

CONFIGURING POWERSQL

PowerSQL uses the PowerSQL configuration table, which consists of the following two tables:

- PowerSQL Control table
- PowerSQL Information table

These tables are configured in the Shared or User domain.

PowerSQL Control Table

PowerSQL is most commonly configured as a User domain task, but it can also be configured as a Shared task.

Accessing

In your server application, open **Data Logging > Power SQL > Power SQL Control**.

Field Descriptions

Control Name Specifies the developer-assigned name of the control record.

Valid Entry: alphanumeric string of 1 to 15 characters

Select Trigger Tag that triggers a select operation. A select operation selects specific data from a relational database table based on information specified in the PowerSQL Information table and places it in a result table for you to view or manipulate.

Valid Entry: tag name

Valid Data Type: digital, analog, longana, float, message

Update Trigger Tag that triggers an update operation. PowerSQL performs a positional update if you defined a Select Trigger. When the value of this tag changes during a positional update, PowerSQL reads the values in the active row (the value of the Current Row tag) and updates the values in that row of the result table and external database.

For a positional update to work, the database table must have a unique index. This can be configured in Database Schema Creation or executed externally to FactoryLink when the database table is created.

PowerSQL performs a logical update if you have not defined a Select Trigger to select specific data. During a logical update, PowerSQL constructs the update SQL statement based on the information entered in the PowerSQL Information table. PowerSQL can process one row or multiple rows of values when the update SQL statement is executed.

To perform an update operation with multiple rows of values, the Current Row Tag and Data Array Size fields must be configured, and the tags in the PowerSQL information table must be tag arrays large enough to hold values determined by the Data Array Size field.

Before setting the Update Trigger, set the Current Row Tag to the number of rows of values that are to be processed by the update statement. The Current Row tag should contain an integer value between 1 and the data array size.

To perform an update operation that processes one row of values, set the Data Array Size field to 1 and leave the Current Row Tag field blank. This configuration causes PowerSQL to use only one row of values when the update operation is executed.

Note: The batch/dynamic mode Update failure, such as duplicate row, will ONLY set the first row's complete status as a proper error code for all historians.

Note: The PowerSQL task supports a feature that permits arrayed operations for updating a supported relational database. Instead of providing a single set of data points to update a single row in a database, this feature uses arrays of data points to perform multiple updates. The batch mode is the most efficient and, at completion, is designed to update an array of status tags for each set of data points (or each operation.) For example, if a batch operation is triggered to "insert" 100 rows using 100 different sets of data with one data set results in a duplicate index key error violation, the status tags should indicate 99 successful inserts and the one error condition of duplicate index key. However, various ODBC drivers behave differently, so check the documentation for the RDBMS and driver you are using.

Valid Entry: tag name

Valid Data Type: digital, analog, longana, float, message

Delete Trigger

Tag that triggers a delete operation. PowerSQL performs a positional delete if you defined a Select Trigger. PowerSQL deletes the active row in the result window from the result table and external database when the value of this tag changes during a positional delete.

The database table must have a unique index for a positional delete to work. This can be configured in Database Schema Creation or when the database table is created.

PowerSQL performs a logical delete if you have not defined a Select Trigger to select specific data. During a logical delete, PowerSQL constructs the delete SQL statement based on the information entered in the PowerSQL Information table. PowerSQL can process one row or multiple rows of values when the delete SQL statement is executed.

To perform a delete operation with multiple rows of values, the Current Row Tag and Data Array Size fields must be configured, and the tags in the PowerSQL information table must be tag arrays large enough to hold values determined by the Data Array Size field. Before setting the Delete Trigger, set the Current Row Tag to the number of rows to be processed by the delete statement. The current row tag should contain an integer value between 1 and the data array size.

To perform a delete operation that processes one row of values, set the Data Array Size field to 1 and leave the Current Row Tag field blank. This configuration causes PowerSQL to use only one row of values when the delete operation is executed.

Valid Entry: tag name

Valid Data Type: digital, analog, longana, float, message

*Insert Trigger Auto Create

This field can be configured as an Insert Trigger or as an auto create field that causes PowerSQL to insert a row in a database table when a logical update operation modifies no rows.

To configure this field as an auto create switch, enter:

YES Insert a new row of data when logical update modifies no rows.

NO Do not insert any new rows. This is the default.

If a tag is configured in this field, it acts as an Insert Trigger. Do not use Select, Update, or Delete Triggers for a control record using an Insert Trigger. The Insert Trigger causes PowerSQL to construct an insert SQL statement based on the information entered in the PowerSQL Information table. PowerSQL can insert one row or multiple rows of data when the insert SQL statement is executed.

Note: The batch/dynamic mode Insert failure, such as duplicate row, will ONLY set the first row's complete status as a proper error code for odbchist/mssql.

To perform an insert operation with multiple rows of values, the Current Row Tag and Data Array Size fields must be configured, and the tags in the PowerSQL information table must be tag arrays large enough to hold values determined by the Data Array Size field. Before setting the Insert Trigger, set the Current Row Tag to the number of rows to be processed by the insert statement. The current row tag should contain an integer value between 1 and the data array size.

To perform an insert operation that processes one row of values, set the Data Array Size field to 1 and leave the Current Row Tag field blank. This configuration causes PowerSQL to use only one row of values when the insert operation is executed.

Valid Entry: YES or NO or tag name
Valid Data Type: digital, analog, longana, float, message

Move Trigger

Tag whose value causes PowerSQL to do a relative move based on the active row. If the Move Trigger contains a negative number, the active row is decreased by this value. If the Move Trigger contains a positive number, the active row is increased by this value. When this operation is completed, the current row tag reflects the position of the active row in the result window. The PowerSQL task scrolls the data rows in the result window to reflect the new position of the active row.

Move operations can be performed only on result tables; therefore, you must have defined and executed a Select Trigger first.

For example, if the value of the Move Trigger tag is 3 and the Current Row tag is 0 (active row is the first row in the result window) and the result window size (data array size) is five rows, the current row tag is changed to 3 and the data in the result window is not scrolled. If the Move Trigger tag is 8, the current row tag is again 3, but the data is scrolled, because the number of rows moved is greater than the result window size.

The scrolling of the data in the result window is controlled by the Move Trigger and by the internal cache size. If the internal cache size is not configured, the active row can only scroll back (Move Trigger is negative) to the row that is at the start of the result window. If the user attempts to scroll back beyond the result window, PowerSQL generates an error and sets the current row tag to 0, because the data that was previously scrolled off the result window was not cached and is no longer accessible by PowerSQL. This configuration does not prevent you from scrolling forward (Move Trigger is positive) to the end of the result table. This configuration is the most efficient, since it uses less memory and disk space to scroll the data in a result window.

Valid Entry: tag name
Valid Data Type: analog, longana

Position Trigger

Tag that moves the result window to the specified row in the result table. The Current Row tag reflects where the active row is positioned within the result window. For example, if the value of this tag is 42, the result window displays row 42 of the result table and the current row tag will reflect where row 42 is in the result window.

Position operations are performed only on result tables; therefore, position operations cannot be performed unless you define and execute a Select Trigger.

The Internal Cache Size field works in conjunction with the Position Trigger tag, also. If the internal cache size is not configured, the Position Trigger tag cannot position the active row to rows that are less than the row at the start of the result window. An attempt to do this causes PowerSQL to generate an error and position the current row tag to 0, because the data that was previously scrolled off the result window was not cached and is no longer accessible by PowerSQL. This configuration does not prevent you from setting the Position Trigger tag to rows that have not yet been displayed in the result window. This configuration is the most efficient, since it uses less memory and disk space to scroll the data in a result window.

Valid Entry: tag name
Valid Data Type: analog, longana

Historian Mailbox Tag used for communication between PowerSQL and a historian. PowerSQL sends requests for information from the relational database to this mailbox tag. The historian task reads this tag and transfers the request to the external database.

Valid Entry: tag name
Valid Data Type: mailbox

***Database.Table Name** Tag or string constant with the database alias name. If a message tag is configured, the user can use this control record to access multiple database tables with like table structures. To use this feature, the message tag must contain the database alias name (defined in the historian task) and the name of the database table that is to be accessed. Place a '.' between the database alias name and the table name. Ensure that this tag contains such a string before a Select, Update, Insert, or Delete Trigger is set.

If the PowerSQL Tag field is configured, only the Database Alias Name is used by PowerSQL and the rest of the string is ignored for this type of operation., because the SQL statement in the PowerSQL tag refers to the database table that is to be accessed.

If the Database.Table Name field references a tag, PowerSQL checks whether this tag has changed since the last select, logical update, logical delete, or insert operation. If it has changed, PowerSQL closes all SQL statements that are referenced by this control record, and creates new ones based on the database table name specified in this tag. For a positional update and a positional delete operation the check is ignored, since these operations are controlled by the Select Trigger operation.

To enter a string constant, use a single quote ‘ as the first character and the database alias name followed by the database table name. Place a “.” between the database alias name and the database table name. If the PowerSQL Tag field is configured, only the database alias name is required.

To fully qualify a database table name, the table name can contain more than one period. Additional periods in the table name must be preceded by the back slash character “\” for PowerSQL to parse this table name correctly.

For example, analias.scott\.mytable. The table name scott\.mytable is fully qualified and requires that the back slash precede the period between scott and mytable. ‘analias’ is the database alias name that is configured in a historian task.

Valid Entry: up to 63 alphanumeric characters or a tag name

Valid Data Type: message

PowerSQL Tag

Tag that is used to supply an SQL statement that PowerSQL executes when either a Select Trigger or Update Trigger is set. PowerSQL reads this tag only when a Select Trigger or Update Trigger is set by the application. Configuring a delete or Insert Trigger is invalid and results in an error at task startup. Only one trigger, a select or update, can be configured when a PowerSQL tag is configured.

Configure an Update Trigger when the SQL statement or stored procedure modifies rows or inserts rows in a database table or drops or creates database objects (tables, indexes, etc.) in a database server. Use a Select Trigger when the SQL statement is a SELECT statement or when a stored procedure returns a result table. If a result table is generated, the user can configure a Move Trigger or Position Trigger. These triggers allow the user to scroll through the result table.

The PowerSQL Tag can contain any valid SQL statement that is valid to the historian task in use. The SQL statement can reference input variables referenced by ‘?’ in the body of the SQL statement. Each input variable must have an associated record in the PowerSQL Information table. The SQL statement can also generate a result table and each result data column must also have an associated record in the PowerSQL Information table. See the description of the Column Expression field in the PowerSQL Information table for more detail. For SQL statements that do not require an input variable or generate a result table, the PowerSQL Information table can be left empty.

Note: You can use only the Select Trigger or Update Trigger to trigger a stored procedure. Do not use the Delete Trigger or Insert Trigger for this purpose. If there is a select statement in the stored procedure, then use the Select Trigger to select the stored procedure; otherwise, use the Update Trigger.

A special syntax is required to have PowerSQL execute a stored procedure. To execute a database-stored procedure, the PowerSQL tag must contain an ODBC-standard escape sequence for executing stored procedures. The ODBC standard escape sequence syntax is

{ [**?=**] **call** *proc-name* [**(** [*parameter*][,[*parameter*]]… **)**] **}**

where

{ }	(Required) brackets begin and end a call statement
?=	(Optional) if stored procedure returns a value and you want it stored in a tag, include this. The ? is a substitution variable (place holder) for the return value.
call	(Required) key word call
proc-name	(Required) name of stored procedure to be executed
( )	(Required) parentheses begin and end the parameter list for a stored procedure.
parameter	list of parameters comma separated. A parameter is a '?' substitution variable or a numeric constant or an SQL string constant.

If the clause is enclosed in [ ], it is optional.

For example,

{ ? = call add_employee(1001, 'John', 'Doe', 'Engineer') }

{ ? = call add_employee(?,?,?,?) }

Note: When using the SQL TAG to execute an SQL statement and the target database is Oracle, (whether using native Oracle historian or ODBC historian,) do not include a ";" at the end of the SQL statement.

Valid Entry: tag name
Valid Data Type: message

Current Row Tag Tag that indicates the position of the active row of data in a result window. After PowerSQL performs a Select, Move, or Position operation, PowerSQL writes the value indicated by the position of the active row to this tag. The value of the current row tag in these operations is between 0 and the data array size – 1.

For select, move, or position operations, PowerSQL writes to the current row tag and the application should treat this tag as read only.

If a Select Trigger is defined, PowerSQL performs all positional update and positional delete operations on the row indicated by the current row tag.

For logical update, logical delete, and insert operations, the Current Row tag value represents the number of rows of values to be processed. The Current Row tag in these operations must be between 1 and the data array size. The values in the array tags that are configured in the PowerSQL Information table must be contiguous, since PowerSQL reads the tag specified in the information table and the next current row tags in the tag array when a Logical Update, Logical Delete, or Insert operation is executed.

Valid Entry: tag name
Valid Data Type: analog

Data Array Size (Rows)

Specifies the number of rows of data contained in a result window or the maximum number of rows of values a logical update, logical delete, or insert operation can process. The tags specified in the Tag Name field of the PowerSQL Information table must be an array large enough to contain values determined by the Data Array Size field. The Data Array Size (Rows) field controls how many rows of data are to be fetched by PowerSQL when a select, move, or position operation is executed.

If the Data Array Size (Rows) field is configured with a number that exceeds the maximum message size that can be processed in a mailbox tag, PowerSQL breaks this operation into several operations until all data rows are processed.

For example, the number specified in this field is 1. Enter a large positive value in the Move Trigger or Position Trigger field to scroll directly to the end of the result table. Because only one row of data is requested at a time, the operation for a large result table takes more time than if the value in this field is larger.

Valid Entry: 1 to 9999

Internal Cache Size (Rows)

Specifies the number of rows of data in a result table that are cached. If the control record does not have a Select Trigger or has a Select Trigger but no Move or Position Trigger, leave this field blank.

If a Select Trigger is defined with a Move or Position Trigger, the value in this field affects the move and position operations.

The internal cache is used to allow the user to scroll forward and backward through the result table that is generated when a Select trigger is executed. If this control record is used to simply load tags with information and only

scrolls forward, the Internal Cache Size field is not necessary and is inefficient for this type of operation. If this control record is used as a table grid for an operator to scroll backward and forward, configure this field so that all rows in a result table are accessed and displayed to the operator.

Observe some guidelines for setting data array size and internal cache size: if this control record is used for an operator viewing a table grid in a graphic screen, do not set the data array size to more than 50, because it is difficult to view more than 50 rows of information in a table grid. A data array size of 50 or less and an internal cache size of 100 provides acceptable performance for operator viewing.

If this control record is used as a way to quickly populate an array of tags that is used to download information from a database table to a PLC, then it makes sense to set data array size to a value larger than 50. For this situation, setting the Internal Cache Size field slows down the operation, since it copies data to memory (twice) and then to disk.

Valid Entry: 0 to 9999

Disable Tag

Tag that disables all related PowerSQL operations.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, or float

Completion Trigger

Tag whose change-status flag is set by PowerSQL when any operation undertaken by this control record is completed.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

Completion Status

Indicates the status of the last operation done by this control record. The completion status tag is updated with status information by PowerSQL. These status messages or status codes are generated by PowerSQL or by the historian task, depending on where the failure takes place. For the codes and messages that can display in this tag, see page 434.

The completion status tag can operate as a single status code or as an array of status codes, depending on the operation executed by PowerSQL. If the completion status tag is a message, PowerSQL updates this tag with a text message. If the completion status tag is an analog tag, this tag displays codes that are described on page 434. If the completion status tag is a longana tag, it displays codes generated by the database server that the historian task is accessing. So these status codes are dependent upon the database server that is connected, and you will need to consult the database server for the definition of the error codes.

If a logical update, logical delete, insert, or SQL operation accepts an array of input values, the status tag can display an array of status codes (only if the completion status tag is either an analog or longana tag type). If an array of status codes is desired, the Completion Status tag must be a tag array and be capable of storing values determined by the Data Array Size field.

You can configure this tag to work in conjunction with output objects in Client Builder to display codes or messages on any graphics screen. You can also configure Math & Logic to monitor this tag and respond to or ignore errors that occur.

Valid Entry: tag name
Valid Data Type: analog, longana, message

Configuration Example

This example assumes the following information is specified in the PowerSQL Control table.

Control Name	Select Trigger	Update Trigger	Delete Trigger	*Insert Trigger Auto Create	Move Trigger
TANK	SELTAG1	UPDTAG1	DELTAG1	‘NO	MRVTAG

Move Trigger	Position Trigger	Historian Mailbox	*Database.Table Name	Dynamic SQL Tag
MVRTAG1	MVATAG1	Histmbx	REFINERY.TANK	

Current Row Tag	Data Array Size (Rows)	Internal Cache Size (Rows)	Completion Trigger	Completion Trigger
CROWTAG1	12	100	COMTRIG1	STATTAG1

In this example, PowerSQL sends a request for select, update, delete, move, and position operations to the historian through the historian mailbox tag HISTMBX. PowerSQL asks for data from the table TANK in the relational database REFINERY.

PowerSQL updates the value of the current row tag CROWTAG1 when PowerSQL performs a select, move, or position operation. The Completion Status tag STATTAG1 contains status information about the operation just completed. The change-status flag for the digital tag COMTRG1 is set when an operation for this result window is complete.

Because the Insert Trigger/Auto Create field indicates NO, PowerSQL does not create a new row and the update operation is not performed whenever you do not find a row for the update operation.

Because the Data Array Size is 12, the result window can display 12 rows of data from the result table at a time. The internal cache can hold 100 rows of data from the result table.

PowerSQL Information Table

Use the PowerSQL Information table to configure the details of the SQL operation defined in the PowerSQL Control table.

Accessing

In your server application, open **Data Logging > Power SQL > "your control name" > Power SQL Information.**

Field Descriptions

Tag Name Tag that contains the values from a column of a relational database table or the values of an SQL expression or the values for input variables for an update, delete, insert, or stored procedure call. If the Data Array Size field in the PowerSQL Control table is greater than 1, the tag must be an array of Data Array Size or greater. Ensure all tags entered in the Tag Name field can accommodate values determined by the Data Array Size field.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

Logical Operator The Logical Operator field is part of a WHERE clause that specifies the conditional statement that restricts the rows selected, updated, or deleted from a database table. The Logical Operator field is ignored for control records that have a PowerSQL tag configured.This field works in conjunction with the Column Expression and Logical Expression fields (described below) to construct the WHERE clause. This can be one of the following:

AND Specifies a combination of conditions in a logical expression.

OR Specifies a list of alternate conditions in a logical expression.

FactoryLink performs a sequential search through the database even if the columns are indexed if you use the OR operator in a logical expression when using the historian for dBASE IV. This may result in a slower response time if the database is large; therefore, we recommend you not use OR operators in logical expressions so the historian for dBASE IV can take advantage of indices.

NOT Negates a condition in a logical expression.

AND_NOT Specifies a combination of conditions and negated conditions in a logical expression.

OR_NOT Specifies a list of alternate negated conditions in a logical expression. (See examples in the following table.)

WHERE clause	Description
Col2 = 3 AND Col4 > 4	PowerSQL selects all rows where Col2 is equal to 3 AND Col4 is greater than 4.
Col3 < 6 OR Col2 >= 19	PowerSQL selects all rows where Col3 is less than 6 OR Col2 is greater than or equal to 19.
Col4 > 7 AND_NOT Col4 = 20	PowerSQL selects all rows where Col4 is greater than 7 AND_NOT equal to 20.

Column Expression Specifies the following items:

(1) A character string representing the relational database column name associated with the Tag Name tag. The Column Name field works in conjunction with the Logical Operator and Logical Expression fields to specify WHERE clauses with the following format:

([table.]column)

where

table is the relational database table name to include if the table name is different from the table name specified in the *Database.Table Name field in the PowerSQL Control table.

column is the name of the column within the relational database table. You can use the same column name in two rows of a table.

OR

(2) An SQL function, such as MAX (col_name) or COUNT (*). The result of this function is written to the tag specified in the Tag Name field. SQL functions are supported only in SELECT statements. SQL functions are not supported in UPDATE statements or by the historian for dBASE IV.

OR

(3) An SQL assignment, such as time_entered=SYSDATE, where time_entered is a column name in a database table and SYSDATE is an Oracle macro that supplies the current time stamp according to the Oracle Server. This feature is valid only for UPDATE and INSERT statements and allows the user to use database server macros or numeric constants or string constants to be entered directly into columns instead of through tags.

OR

(4) The reserved words $OUTPUT, $INPUT, and $INOUT. These reserved words are valid only for control records that have a PowerSQL tag configured in the control record. The reserved words tell PowerSQL how to treat the tag referred to by Tag Name and the associated SQL statement in the PowerSQL Tag message tag.

If Column Expression is $OUTPUT, the tag in the Tag Name field holds values for a column in a result table generated by either a SELECT statement or a stored procedure. If the Column Expression is $INPUT, the tag in the Tag Name field holds the value for a substitution variable '?' embedded in the body of the SQL statement. If the Column Expression is $INOUT, the tag in the Tag Name field holds the value for a substitution variable in a stored procedure call statement and also updates the tag with the value that is returned by the stored procedure in this variable.

Valid Entry: 63 alphanumeric characters

Maximum Character Data Size

The maximum size in bytes that PowerSQL will write to a message tag. This field is supplied because SQL expressions that result in a character string may default to a large data size that will cause excessive memory in the FactoryLink real time database to be allocated and wasted. If this field is left blank, PowerSQL always writes to the message tag the default size that is supplied by the database server for the associated column expression.

Valid Entry: 1 to 256

Logical Expression

Used to generate a conditional statement that restricts the rows selected, updated, or deleted from a database table. This field works in conjunction with the Column Expression and Logical Operator fields to generate the WHERE clause used in the SQL statement. The Logical Expression field is ignored for control records that have a PowerSQL Tag configured.

Note: An embedded variable in PowerSQL is a FactoryLink tag name preceded by a colon. The embedded variable can be used only in the Logical Expression field. The embedded variable can be any FactoryLink tag type except mailbox. If the tag is an array, specify the dimension (for example, :tag_xyz[2]). The tag in the embedded variable is not detected by Configuration Explorer as a tag, so the user must define the tag somewhere else in the application, such as in Math & Logic.

The conditional statement in a Logical Expression field can consist of relational operators, such as =, <, >=, and others. Consult the RDBMS SQL Language user's manual for supported relational operators for the historian in use.

The WHERE clause is generated by appending the Logical Operator, Column Expression, and Logical Expression fields in the order displayed in the PowerSQL Information table. Punctuation is supplied by PowerSQL to ensure correct SQL syntax. Any embedded variable found in the Logical Expression field is replaced by a '?', which SQL defines as a substitution variable for a value to be supplied at execution time. The value supplied is the tag's value defined by the embedded variable.

The string generated by this is a WHERE condition. If the first word(s) in this string is not SQL-reserved words such as ORDER BY, the reserved word WHERE is attached to the start of this string. The user must ensure that any placement of SQL clauses such as ORDER BY and GROUP BY is properly ordered as defined by the SQL language for the targeted database server.

The ORDER BY clause is supported in the dBASE IV historian, but only to the extent that the columns listed in the ORDER BY clause must match an index that was created for the database table. The dBASE IV historian does not build any temporary tables to reorder the rows, so make sure the ORDER BY clause matches an index for the dBASE IV database table. If an ORDER BY clause does not match an index, the dBASE IV historian returns an error.

If you define a Select Trigger in the PowerSQL Control table, the WHERE clause is used for the select statement. If a Select Trigger is not defined, the WHERE clause is used for either the logical update operation or logical delete operation or for both.

A logical expression can contain one of the following:

1. Character string of up to 79 characters containing an SQL expression:

OUTLETTVAL = 30 and TANKID = 'BLUE001'

or an SQL clause:

ORDER BY TANKID

2. Character string of up to 79 characters representing an SQL expression that contains embedded variables. If the tag is a message tag, the character data in the message tag should not be enclosed in single quotes. If the PowerSQL Control record has no Select Trigger configured and the data

array size is greater than one, the tags referenced by the embedded variables must be tag arrays large enough to contain values determined by the Data Array Size field.

For example:

=:tagTANKID

where tagTANKID is a message tag of value: *BLUE001*

3. An embedded message variable, which must be a message tag. The message tag contains an SQL clause or SQL expression. The SQL expression cannot contain an embedded variable and any string constants in the SQL expression must be quoted in single quotes.

For example:

:tagSQLExpression

where

tagSQLExpression is a message tag

OUTLETVAL = 30 and TANKID = 'BLUE001'

Note: Options 1 and 3 are different. The result is the same for both options, but option 3 allows the user to change the SQLExpression tag to a different expression before setting a Select, Update, or Delete Trigger, thereby altering the rows selected, updated, or deleted. Option 1 is always static and cannot be changed at run time. Option 2 allows the user to change the value of tagTANKID, but the SQL expression is still the same. Only the search criterion for the WHERE clause has changed.

PowerSQL substitutes embedded variables with the value of the tag defined in the embedded variable when executing the select, update, or delete SQL statement.

For example:

=:tagTANKID

generates the following WHERE clause:

where TANKID = ?

TANKID is the value of the Column Name field.

PowerSQL reads the value of the tag tagTANKID from the real-time database and substitutes its value for the '?' when it executes a select, update, or delete SQL statement.

Configuration Example

This example uses the following information in the PowerSQL Information table.

Tag Name	Logical Operator	Column Expression	Maximum Character Data Size	Logical Expression
TANKID[3]		TANK.TANKID	33	=’BLUE001”
outlet[3]	AND	TANK.OUTLET	0	>=:OUTLETVAL

Because the Select Trigger tag SELTAG1 (defined in the Control table) is digital in this example, the historian returns the two following values to PowerSQL when the change-status flag for SELTAG1 is set:

- Values where the column named TANKID equals BLUE001
- The column named OUTLET is greater than or equal to the value of the tag OUTLETVAL.

PowerSQL writes these values to the tags contained in the tag arrays TANKID[3] and OUTLET[3]. These values are then displayed in a result window.

Each Tag Name tag displays one column of values in a result window. Because an array has been defined for TANKID and OUTLET, the values in the columns the for which the logical expression is true are displayed in the result window.

Stored Procedure Example for Oracle

```
-- echo Building Oracle Package PKGCSP03...
drop package PKGCSP03;
Create or replace package PKGCSP03 as
type char_array is table of varchar2(10)
index by binary_integer;
    type int_array is table of integer(10)
index by binary_integer;
procedure updsel_trendtbl (
inrecs in integer,
key in integer,
newtime in string,
addsec in integer,
outtime out char_array,
outsec out int_array,
outrecs in out integer);
```

```
End PKGCSPO3;
/
Create or replace package body PKGCSPO3 as
cursor c1 (key in integer) is
select fltime, flsec from trendtbl where trendkey >= key;
procedure updsel_trendtbl (
inrecs in integer,
key in integer,
newtime in string,
addsec in integer,
outtime out char_array,
outsec out int_array,
outrecs in out integer) is
begin

update trendtbl set
fltime = newtime,
flsec = flsec + addsec
where trendkey = key;
commit work;
if c1%ISOPEN, then
    close c1;  -- close cursor if it is open
end if;
open c1(key);  -- open cursor
outrecs: = 0;  -- init rows found
for i in 1..inrecs loop
    fetch c1 into outtime(i), outsec(i);
    if c1%NOTFOUND, then
close c1;  -- close cursor if no rows found
exit;
    else
outrecs: = outrecs + 1;  -- count row found
    end if;
end loop;
end updsel_trendtbl;
End PKGCSPO3;
/
commit;
```

Note: Tecnomatix is not responsible for any changes in Oracle. Refer to the Oracle manual for any changes.

PROGRAM ARGUMENTS

Argument	Description
-C<#> or –c<#>	In earlier versions of PowerSQL, a COMMIT statement was performed after all database accesses, (except the SELECT statement), and were executed as a nondynamic SQL statement. The execution of dynamic SQL statements, especially for stored procedures, can result in complex database operations that include many steps. In such cases, PowerSQL cannot determine if a COMMIT or a ROLLBACK is more appropriate. This has the potential to COMMIT unwanted database updates in the case of execution failures. Proper procedures dictate that COMMIT/ROLLBACK logic should be programmed into the stored procedures. However, since changing how this works might have an impact on an existing application, the task has been modified to accept a program argument that controls the COMMIT logic. (# = 0, 1, or 2) **-c1** results in no COMMITs for dynamic SQL statements. The nondynamic SQL operations (traditional insert, delete, and update statements) are followed by a COMMIT. **-c2** (the default action) is to COMMIT logic exactly as in the earlier version, so no modifications are required to existing applications. However, it is strongly recommended that the applications be modified to use the **-c1** argument and that all stored procedures be updated to include all necessary and appropriate COMMIT/ROLLBACK logic. **-c0** results in no COMMITs for any statements executed, except for a final COMMIT upon task shutdown. Use of “-c0” is not recommended, since failure to COMMIT nondynamic SQL statements could have an adverse effect on the database server, but the configuration is included for completeness. Since a COMMIT can be easily executed through the use of the SQL tag, it allows users to take responsibility for COMMIT logic away from PowerSQL and make it become part of the application design and control.

Argument	Description
-L or –l	Enables logging of errors to the log file. By default, PowerSQL does not log errors.
N or –n	Notifies on the completion of a SELECT trigger that the query resulted in an End of Fetch condition. Notification will only occur if the rows returned from the query do not equal the rows defined in the Data Array Size field. By default, PowerSQL does not report an End of Fetch condition for a SELECT until a move operation advances the current row past the last row of the query.
-S<#> or –s<#>	Sets the maximum number of SQL statements that PowerSQL will have active at one time. The default is 160. For very large applications, this program switch may have to be adjusted if the database server is unable to allocate a resource to open a new SQL cursor. (# = 4 to 60)
-W<#> or -w<#>	Sets the maximum timeout in seconds for PowerSQL to wait for a response from the historian task. The default is 30 seconds. (# = 5 to 36000)
-V1 or -v1	Writes the SQL statements generated by PowerSQL to the log file. PowerSQL must have logging enabled for this program switch to work. The default is to not write the SQL statements to the log file.

STATUS CODES AND MESSAGES

This section describes the error messages that can be written to the Completion Status tag or displayed on the Run-Time Manager screen for the PowerSQL task. The codes and messages can also be viewed on a graphics screen if you define an output text object to display them.

Completion Status Messages

Code	Description	Cause and Action
100	Asynchronous error from historian function.	Cause: An SQL COMMIT operation failed within the historian. Action: Consult the database administrator for the external database in use.
101	Error from historian function.	Cause: A syntax error may have been made or information may not have been entered in a required field in a configuration table. Action: Correct the SQL statement syntax error by modifying the Information table for the task receiving the error. If the information is correct, ensure the database table exists.
102	No fields for select.	Cause: The task is trying to execute a select operation, but no tag names have been defined to hold the data from the select operation. Action: Define some tag names in the Tag Name field in the PowerSQL Information table.
103	No fields for insert.	Cause: The task is trying to execute an insert operation, but no tag names have been defined to hold the data from the insert operation. Action: Define some tag names in the Tag Name field in the PowerSQL Information table.
104	No fields for update.	Cause: Task is trying to execute an update operation, but no tag names have been defined to hold the data from the update operation. Action: Define some tag names in the Tag Name field in the PowerSQL Information table.

Code	Description	Cause and Action
105	Update and delete operations not supported with multi-table view.	Cause: Update and delete operations cannot be performed when using multi-table view. Action: Do not perform update and delete operations when using multi-table view.
106	Cannot update until select is performed.	Cause: A select trigger is defined, but a select operation has not executed. A select operation must be executed before an update operation. Action: Execute a select operation and then retry the update operation.
107	Cannot delete until select is performed.	Cause: A select trigger is defined, but a select operation has not been executed. A select operation must be executed before a delete operation. Action: Execute a select operation and then retry the delete operation.
108	Cannot move until select is performed.	Cause: A select trigger is defined, but a select operation has not been executed. A select operation must be executed before a move operation. Action: Execute a select operation and then retry the move operation.
109	This row of data has been deleted.	Cause: A delete operation attempted on a nonexistent row. Action: No action required.
110	A FactoryLink function returned an error.	Cause: A PAK function encountered an unknown or unexpected error. Action: Contact your technical support representative.
111	A file function error occurred.	Cause: System encountered an unknown error while trying to read or write to the external buffer. Action: If there is not enough disk space, decrease the buffer size.
112	Bad tag in logical expression.	Cause: Either a typographical error exists or an undefined or invalid tag name is entered as an embedded variable tag in a Logical Expression field. Action: Correct typographical errors. If you did not make a typographical error, then define the tag.

Code	Description	Cause and Action
113	Invalid use of tag in logical expression.	Cause: Logical expression does not contain a valid tag name. Action: Correct typographical errors and ensure the tag name is the name of a valid tag.
114	HSDA structure too small.	Cause: Invalid use of a tag in logical expression. Action: Define the tag used in the logical expression.
115	Cannot open log file.	Cause: Disk space too low. Action: If disk space is low, delete unneeded files or programs.
116	A request for memory failed.	Cause: Internal error. Action: Contact your technical support representative.
117	Cannot find unique index for table.	Cause: A positional update or delete operation occurred on a table without a unique index. Action: Create a unique index for the table and retry the operation.
118	PowerSQL Information record has an invalid configuration.	Cause: The information record has an assignment statement, and the logical operator and/or logical expression are configured. The assignment statement references either a numeric constant or a string literal, and a tag is configured in the Tag Name field. The assignment statement has a substitution marker with no associated tag in the Tag Name field. Action: Change the information record.
119	Tag array is too small for PowerSQL operation.	Cause: A tag referenced in the information table is not large enough to contain Data Array Size values. Action: Change the dimension for the tag or enter another tag large enough to contain Data Array Size values.

Run-Time Manager Messages

The first three letters shown in the messages below as *nnn* are a variable that indicates whether the message came from PowerSQL or from the Historian (HIS).

Error Message	Cause and Action
nnn*-[BAD_SMBX] Bad send mailbox. Control name: *name	Cause: You entered the wrong mailbox tag name. Action: Look up the mailbox tag name for the historian being used in the Historian Mailbox field of the Historian Mailbox Information table. Enter the correct name in the Historian Mailbox field of the PowerSQL Control table.
nnn*-[BAD_WHERE_TAG] Bad tag name for logical expression. Control name: *name	Cause: Either you made a typographical error or entered an undefined or invalid tag name as the embedded message tag in a logical expression. Action: Correct all typographical errors. Define the tag if it does not exist.
nnn*-[CT_HDR] No sel, upd, or delete trigger defined. Control name: *name	Cause: The Select, Delete, or Update trigger is not defined. Action: Define at least one of the triggers listed above.
nnn*-[DBTBL_SYNTAX] The Database Table value is missing a '.'. Control name: *name	Cause: You left the '.' out of the entry in the Database Table Name field in the PowerSQL Control table. Action: Put a '.' between the database name and the table name in the entry in the Database Table Name field.
nnn*-[FL_FUNC] Function 'function' returned error *error code*. Control name: *name	Cause: The PAK function encountered an unknown or unexpected error. Action: Contact your technical support representative.
nnn*-[FL_FUNC] Function 'FL_WRITE' returned error 9. Control name: *name	Cause: The column type in the database that FactoryLink is trying to read from does not match the column type defined in the Database Logging task Schema Creation table. Action: Redefine the column type in the Column Type field of the Schema Creation table to be the same as the column type in the database.
***nnn*-[HSCONNECT] Failed to connect to Historian**	Cause: You may have specified the wrong mailbox tag name. Action: Specify the historian predefined mailbox tag name in the Historian Mailbox field in the PowerSQL Control table.

Error Message	Cause and Action
nnn*-[HSDA_TOO_SMALL] HSDA structure too small. Control name: *name	Cause: You specified an invalid tag in a logical expression. Action: Define the tag used in the logical expression in a FactoryLink task other than PowerSQL. The tag will then be valid in the logical expression.
nnn*-[HSDBERROR] Historian database error: *error message	Cause: This message is accompanied by various other messages that describe the cause of the error. Action: Read the accompanying message displayed on the Run-Time Manager screen or in the .LOG file and correct the problem based on the instructions.
***nnn*-[HSDUPLICATE] Tried to insert a duplicate row**	Cause: You tried to insert a duplicate row into a result table. Action: No action required.
***nnn*-[HSENDOFETCH] Last row fetched or row not found**	Cause: The task could not find a row during an update operation because that row does not exist. Or, during a move or position operation, you specified a nonexistent row (for example, row 100 when the table has only 50 rows). You attempted to go past the end or above the beginning of the result table. Action: No action required.
***nnn*-[HSFLDEXISTS] Tried to add an existing field**	Cause: You tried to add an existing field. Action: No action required.
***nnn*-[HSMAXOPENS] Too many open sessions**	Cause: You tried to open data from more than ten unique databases. Action: Reference fewer than 10 unique databases in a configuration table.
***nnn*-[HSMEMORY] Memory error malloc failed**	Cause: Not enough memory is allocated for the historian. Action: Allocate more memory for the historian.
***nnn*-[HSNOFIELD] Tried to access a nonexistent field**	Cause: You tried to open a nonexistent field. Action: No action required.
***nnn*-[HSNOTABLE] Tried to access a nonexistent table**	Cause: You tried to open a nonexistent table. Action: No action required.

Error Message	Cause and Action
***nnn*-[HSPREPARE] Failed to prepare stmtid**	Cause: A nonexistent table name or field name is specified or a syntax error is made in an SQL statement. Action: Ensure all entries in the PowerSQL table are correct, especially those for the SQL statement.
***nnn*-[HSSTMTID] Invalid stmtid returned from Historian**	Cause: The historian shut down before PowerSQL or another historian-client task. Action: Shut down PowerSQL and all other historian-client tasks running on the system. Then, shut down the historian and restart it, followed by PowerSQL and all other historian-client tasks.
***nnn*-[HSTBLEXISTS] Tried to create an existing table**	Cause: You tried to create an existing table. Action: No action required.
***nnn*-[HSTIMEDOUT] Historian not responding. Maximum timeout exceeded.**	Cause: A PowerSQL request did not get a response from historian within the timeout period. Action: If the timeout period is less than the time taken to serve the request, you may increase the timeout, e.g., from -w300 to -w400.
***nnn*-[HSUNKNOWN] Unknown function request sent to Historian**	Cause: An error occurred within PowerSQL. Action: Contact your technical support representative.
nnn*-[INVUSE_WHERE_TAG] Invalid use of tag in logical expression. Control name: *name	Cause: A logical expression contains an invalid tag name. Action: Correct all typographical errors and ensure the tag name is valid.
nnn*-[MULTI-VIEW] Update and delete operations not supported with multi-table view. Control name: *name	Cause: You tried to perform an update or delete operation while using multi-table view. Action: Do not try to perform update or delete operations.
nnn*-[NO_FLDS_INS] No fields for insert. Control name: *name	Cause: The task is trying to perform an insert operation but no tag names are defined to hold the data from the insert operation. Action: Define some tag names in the Tag Name field of the PowerSQL Information table.

Error Message	Cause and Action
nnn*-[NO_FLDS_SEL] No fields for select. Control name: *name	Cause: The task is trying to execute a select operation but no tag names are defined to hold the data from the select operation. Action: Define some tag names in the Tag Name field of the PowerSQL Information table.
nnn*-[NO_FLDS_UPD] No fields for update. Control name: *name	Cause: The task is trying to execute an update operation, but no tag names have been defined to hold the data from the update operation. Action: Define some tag names in the Tag Name field of the PowerSQL Information table.
nnn*-[NO_LOGICAL_EXPR] No logical expr. Control name: *name*. Column name: *name	Cause: A logical operation was defined, but the logical expression was not specified. Action: Either delete the operation or create a logical expression.
***nnn*-[NO_MEMORY] Out of RAM**	Cause: Not enough RAM is available to run this task. Action: Allocate more RAM for the PowerSQL task.
nnn*-[NOTASSOC] Col name *name* not associated with Tag Name or Logical Expression. Control name: *name	Cause: A nonexistent or invalid tag name is specified. A column name is also specified, but not a logical expression. Action: Define a tag in the Tag Name field of the PowerSQL Information table and/or specify a logical expression.
nnn*-[NO_UNIQ_INDX] Cannot find unique index for table. Control name: *name	Cause: You attempted a positional update or delete operation on a table without a unique index. Action: Create a unique index for the table and retry the operation.
nnn*-[NULL_ROW] This row of data was deleted. Control name: *name	Cause: You attempted a delete operation on a deleted row. Action: No action required.
nnn*-[NULL_TABLE] No data for this table. Control name: *name	Cause: You tried to perform an update, move, or delete operation on a result table that contains no rows of data for either of two reasons: the select operation resulted in no rows of data or you deleted all rows of data from the table. Action: No action required.
***nnn*-[OPEN_LOG] Cannot open .LOG file.**	Cause: The computer may have run out of disk space. Action: Delete any unnecessary files or programs.

Error Message	Cause and Action
nnn*-[SEL_B4_DEL] Cannot delete until select is performed. Control name: *name	Cause: A select trigger is defined, but a select operation was not executed. A select operation must be executed before a delete operation can be performed. Action: Execute a select operation and retry the delete operation.
nnn*-[SEL_B4_MOVE] Cannot move until select is performed. Control name: *name	Cause: A select trigger is defined, but a select operation was not executed. A select operation must be executed before a move operation can be performed. Action: Execute a select operation and then retry the move operation.
nnn*-[SEL_B4_UPD] Cannot update until select is performed. Control name: *name	Cause: A select trigger is defined, but a select operation was not executed. A select operation must be executed before an update operation can be performed. Action: Execute a select operation and retry the update operation.
nnn*-[SQL_ASYNC] Asynchronous failure to name. Error: *error message	Cause: An SQL COMMIT operation failed within the historian. For Oracle, it can fail if you do not have enough disk space. Action: Consult the administrator for the database in use.
nnn*-[SQL_SYNC] Historian function *function* failed. Error: *error message* Control name: *name	Cause: A syntax error may have been made or there may not be any information in a required field in a configuration table. Action: Modify the information in the PowerSQL Information table to create a correct SQL statement if the error is a syntax error. Ensure that the database table exists if the tables are correct.
nnn*-[UNSOL_MSG_RCVD] Unsolicited message received from *tag number	Cause: Another task wrote to PowerSQL's mailbox tag (PowerSQL is not expecting to hear from this task). Action: Determine which task is writing to PowerSQL's mailbox tag by looking at the X-reference list. Correct the problem by changing the mailbox tag name of the task writing to PowerSQL's mailbox tag.
F_BAD_CT_SIZE Bad size for CT # *CT number*	Cause: There is a discrepancy between the PowerSQL script file and the size. You may have modified the PowerSQL .CTG file. Action: Copy the .CTG file from the Installation disk over the modified one.

Error Message	Cause and Action
F_BAD_RMBX Invalid global mailbox: *name*	Cause: PowerSQL's predefined mailbox tag is nonexistent in the GLOBAL.CT file. Action: Contact your technical support representative.
F_INIT Task initialization failed	Cause: This message is preceded by another error message that explains the cause of the error. Action: Examine the preceding message to determine the cause of the initialization failure.
F_NO_CTS No valid tables in CT archive *file name*	Cause: The PowerSQL table has been configured. Action: Configure the PowerSQL table.
F_NO-DOMAIN_NAME No domain name for appl. directory *directory*	Cause: No domain name is specified for the application directory. Action: Specify a domain name for the application directory.
F_OPEN_CT Cannot open CT archive *file name*	Cause: A .CT file may have been deleted. Action: Use CTGEN to rebuild the .CT file.
F_READ_CT Cannot read CT # *CT number* in CT archive *file name*	Cause: A .CT file is corrupt. Action: Rebuild the corrupt .CT file.
***nnn*-[HSENDOFETCH] Tried to move beyond the end of result table**	Cause: A move operation tried to place current row to a row beyond the result table. Action: No action required.
***nnn*-[HSENDOFETCH] Tried to move beyond the top of result table or view**	Cause: A move operation tried to place current row to a row beyond the result table. Action: No action required.
***nnn*-[HSENDOFETCH] Move operation failed due to an empty result table**	Cause: Cannot perform move operation on an empty result table. Action: No action required.
***nnn*-[HSENDOFETCH] Absolute move did not move to requested row because it is deleted**	Cause: Absolute move cannot set active row to a deleted row. Action: No action required.
***nnn*-[HSENDOFETCH] Move operation failed because all rows deleted in desired direction**	Cause: Cannot move in a certain direction because all rows have been deleted in that direction. Action: No action required.

Error Message	Cause and Action
nnn*-[INCORRECT_MODE] Record *recnumber* mode type does not match SQL operation. Control name: *name	Cause: The SQL statement in Dynamic Tag does not match the mode type for the PowerSQL information record. It must be a type of $INPUT, $OUTPUT, or $INOUT. Action: Change the SQL statement in PowerSQL tag or change the information record to the correct mode.
nnn*-[INPUTROWS] Input rows must be between 1 and Data Array Size. Control name: *name	Cause: The current row tag must be between 1 and Data Array Size. Action: Set current row tag to a value between 1 and Data Array Size and then retry the operation.
nnn*-[DYNAMIC_COLUMNS] Only $OUTPUT, $INPUT, $INOUT column expressions allowed. Control name: *name	Cause: A Column Expression field in information table must contain $OUTPUT, $INPUT, or $INOUT reserved words to execute SQL statement that is in the PowerSQL tag. Action: Change the column expression field to $OUTPUT, $INPUT, or $INOUT.
nnn*-[SQLTRIGGER] Only SELECT or UPDATE trigger allowed. Control name: *name	Cause: Only a SELECT or UPDATE trigger can be configured to execute a statement contained in a PowerSQL Tag. Action: Configure a SELECT or UPDATE trigger.
nnn*-[INSERT_TRIG] SELECT or DELETE trigger not allowed. Control name: *name	Cause: When an INSERT trigger is configured, a SELECT and/or DELETE trigger cannot be configured. Action: Remove the SELECT and/or DELETE trigger in the control record.
nnn*-[SQLEMPTY] SQL message tag is an empty string. Control name: *name	Cause: PowerSQL tag is empty. Action: Set the PowerSQL tag with a valid SQL statement.
nnn*-[DESCRIBE] Only one $OUTPUT record allowed. Control name: *name	Cause: To use the DESCRIBE TABLE statement, only one information record is allowed, the column expression must be $OUTPUT, and the Tag Name must reference a message tag. Action: Change information record column expression field to $OUTPUT and ensure that a message tag is placed in the Tag Name field.
nnn*-[ARRAY_TOO_SMALL] TAG array in record *recnumber* is too small. Control name: *name	Cause: The tag array is too small based upon the Data Array Size field value. Action: Enlarge the tag array to ensure that the Data Array Size values can be stored in the tag array.

Error Message	Cause and Action
nnn*-[TAG_TOO_SMALL] TAG *tagname* dimensions too small. Control name: *name	Cause: The dimensions of the tag that is referenced in an information record Tag Name or Logical Expression field is not large enough to store Data Array Size values. Action: Change the dimensions of the tag to ensure the Data Array Size values that can be stored in the tag array.
nnn*-[DESCRIBE_TAG] Message TAG type required for output of a DESCRIBE TABLE statement. Control name: *name	Cause: To use the DESCRIBE TABLE statement, only one information record is allowed, the column expression must be $OUTPUT, and the Tag Name must reference a message tag. Action: Change information record column expression field to $OUTPUT and ensure that a message tag is placed in the Tag Name field.
nnn*-[INVTAG_SYNTAX] Invalid tag syntax: *tagname	Cause: Incorrect tag name syntax entered in the Logical Expression field of information record. Action: Enter a valid tag name.
***nnn*-[NO_OBJ_CT] Cannot access OBJECT CT**	Cause: Internal error; cannot access the Tag Name database. Action: Contact your technical support representative.
nnn*-[NO_FIND_TAG] Cannot find tag: *tagname	Cause: The tagname referenced in the Logical Expression field of information record cannot be found. Either the tag is misspelled, or the tag has not been defined somewhere else in the application. Action: Check spelling or define the tag.
***nnn*-[TAG_WRONG_DOMAIN] Tag *tagname* cannot be referenced by domain *domain type*.**	Cause: The tagname referenced in the Logical Expression field of information record cannot be accessed by PowerSQL task because it is in the wrong domain. Action: Edit the tag so that it is accessible to PowerSQL, or enter a new tag with the correct domain.
nnn*-[INV_DIM_SPEC] Invalid dim specifiers for tag: *tagname	Cause: The tagname referenced in the Logical Expression field of information record has an invalid dimension specifier. Action: Ensure that the dimensions of the tag are correct based on the tag definition.

Error Message	Cause and Action
nnn*-[UPDCOL_EXPR] Column assignment *column expression* cannot have other fields configured. Control name: *name	Cause: 1) The Logical Operator or Logical Expression fields in the information record cannot be configured, or 2) The column assignment expression references a substitution variable "?", and no Tag Name field is configured, or 3) The Tag Name field is configured but the column assignment expression in the Column Expression field does not contain a substitution variable "?". Action: Ensure that the Logical Operator and Logical Expression fields are empty and make sure Tag Name field is correct for the column assignment expression entered into the Column Expression field.
nnn*-[FL_FUNC] Error reading tag. Error = *error code*. Control name: *name	Cause: The fl_read FactoryLink PAK function call encountered an unknown or unexpected error. Action: Contact your technical support representative.
nnn*-[INPUTS_TOO_SMALL] Not enough input records configured for SQL operation. Control name: *name	Cause: The PowerSQL tag in the control record references more input variables than what has been configured in the PowerSQL information table. Action: Add more input records to the PowerSQL Information table or change the SQL statement in the PowerSQL tag.
nnn*-[OUTPUTS_TOO_SMALL] Not enough output records configured for SQL operation. Control name: *name	Cause: The PowerSQL tag in control record references more output result columns than what has been configured in the PowerSQL information table. Action: Add more output records to the PowerSQL Information table or change the SQL statement in the PowerSQL tag.
nnnn*-[INPUTS_UNEQUAL] The number of input records does not match SQL requirements. Control name: *name	Cause: The SQL statement generated by PowerSQL does not match what is configured in the PowerSQL Information table. Action: Contact your technical support representative.
nnn*-[OUTPUTS_UNEQUAL] The number of output records does not match SQL requirements. Control name: *name	Cause: The SQL statement generated by PowerSQL does not match what is configured in the PowerSQL Information table. Action: Contact your technical support representative.

Chapter 18. . . .
Print Spooler

The FactoryLink Print Spooler allows you to direct data to printers or other devices with parallel interfaces and also to disk files. The Print Spooler task also provides other features:

- File name spooling (loads file when print device is available, minimizing required memory)
- Management of printing and scheduling functions

Print Spooler receives output from other FactoryLink tasks, such as Alarm Supervisor or File Manager, and sends this output to a printer or disk file.

With Print Spooler, you can define up to five devices to receive output from other FactoryLink tasks. To send files to one of these devices, FactoryLink tasks reference the corresponding device number in a configuration table.

PRINT SPOOLER INFORMATION TABLE

Use the Print Spooler Information table to specify the devices where the Print Spooler task directs data.

Accessing

In your server application, open **Reports > Print Spooler > Print Spooler Information**.

Field Descriptions

Device — Name of the output device. Each line corresponds to a specific device number. For example, line1 = device 1 and line 5 = device 5. With Print Spooler, you can define up to five devices (lines) to receive output from other FactoryLink tasks.

You can assign two or more device numbers to the same physical device. For example, if only one printer is installed and it is attached to parallel port 1, you can enter the same device name for both Device 1 (line 1) and Device 2 (line 2). Print Spooler then recognizes Devices 1 and 2 are the same physical device and it acts accordingly.

FactoryLink tasks use the first entry in the Print Spooler Information table as the default output device. For example, when you request a print screen in Run-Time Graphics, the output goes to the first device defined in the Print

Spooler table. You can change this default by moving the information for another defined device to the table's first line using the cut and paste features.

You can also direct output to a file rather than to a port by specifying a path and file name. If Print Spooler writes to an existing file, the new values/text are appended to that file in the specified format.

If output is redirected to a disk file as opposed to a true device, the file opens in append mode before it receives output. If the file does not exist when Print Spooler is first started, it is created if any print jobs are directed to it. The characters written to it are precisely the same as if it were a true device, including all device command sequences. You can use such file redirection to capture output for later examination and/or printing at a time when FactoryLink is shut down.

The Print Spooler task can send output to multiple devices at once.

The following example contains valid entries:

Operating System	Printer Specifications	Path and File Name Format
Windows 2000	lpt1, COM1 lpt2, COM2 lpt3	DRIVE:\DIRECTORY\ SUBDIRECTORY\FILE.EXT

Initialization Sequence/File Separator Sequence

The Initialization Sequence is defined by entering the action you want it to perform in the Initialization Sequence column of the Print Spooler Information table. The Initialization Sequence performs the action(s) you define once at the beginning of a Spooler session.

The File Separator Sequence is defined by entering the action you want it to perform in the File Separator Sequence column of the Print Spooler Information table. The File Separator Sequence performs the action(s) you define at the end of each file of a Spooler session.

Enter a sequence for use only with the Report Generator and File Manager, not for use with the Alarm Supervisor command sequences that automatically send characters to the printers to separate the output of different files.

These command sequences can consist of two types of characters:

Display characters—Printable ASCII characters, such as A, that can be printed between files.

Control characters—Codes that instruct a printer to perform an action. Control characters have ASCII values from 00 through 1F hexadecimal. Enter a backslash (\) followed by the two-character hexadecimal value of the control character to place a control character in a sequence.

For example, enter the command sequences as follows (form feed has the ASCII hex value 0C) to include a form feed command between files:

Initialization Sequence:'\0C'
File Separator Sequence:'\0C'

The backslash character itself can be entered as two backslashes (\\).

For information about control characters and their hexadecimal values, refer to the user manual for the appropriate printer.

Valid Entry: up to 16 alphanumeric characters

Binary On/ Binary Off

Specifies command sequences sent to the printers to print binary files. Only for use with the Report Generator and File Manager; not with the Alarm Supervisor. These sequences represent ASCII control characters as two-digit hex values. These command sequences can consist of two types of characters:

Display characters—Printable ASCII characters, such as A, that can be printed between files

Control characters—Codes that instruct a printer to perform an action. Control characters have ASCII values from 00 through 1F hexadecimal. Enter a backslash (****) followed by the two-character hexadecimal value of the control character to place a control character in a sequence. For example, the two-digit hex value for the ASCII ESCAPE character is 1B. If, on a particular printer, an ESCAPE character followed by G turns graphics mode ON and ESCAPE followed by g turns it OFF, fill in the command sequences as follows:

Binary On:\1BG
Binary Off:\1Bg

This sequence puts the printer in graphics mode before it begins the print job and takes the printer out of graphics mode upon completion of the job.

Valid Entry: up to 16 alphanumeric characters

Status Tag

Tag updated by Print Spooler that contains one of three analog values representing the status of the printing device:

0 File print is complete

1 Device active

2 Error operating device

Be sure to distinguish between devices when assigning tag names if you specify more than one device.

Valid Entry: tag name
Valid Data Type: analog

Message Tag

Tag that contains a message describing the status of the printing device. This tag is useful if you want to display the status of a printing device on a graphics screen. To display the status of a printing device on a graphics screen, configure an output text object in the Client Builder to contain the value of this tag. Be sure to distinguish between devices when assigning tag names if you specify more than one device.

Caution: All Print Spooler job requests are routed through the **/FLAPP/SPOOL** directory. If all such requests are not effectively transferred (for instance, if the printer is offline), this directory can get backlogged. To ensure effective processing, check the directory periodically and delete any obsolete job requests.

Valid Entry: tag name
Valid Data Type: message

Use OS Print Services

Enables the spooler to use a print services configured network printer. (When submitting print jobs through the OS print services, ESCAPE sequences cannot be imbedded in the jobs.)

A Print Services dialog was added to allow FactoryLink print spooler to run through the operating system print server.

If Print Services is set to NO the Print Spooler writes directly to the printer. The additional fields, font name and font size, are ignored.

If Print Services is set to YES, the following points apply:

Print spooler does not support DOS-type print capture mapping. Network printers need to be specified by name or by network device.

Print spooler supports connection to printers by printer name or port name, where the port name is the actual physical port name of the printer; for example, LPT1 is the printer connected to the LPT1 port.

Because multiple printers can be configured to use the same port, the spooler reports an error if a printer port has multiple printers assigned.

Printer Font

Defines the font name to use for printing. This must be a valid printer font. Refer to your printer's documentation for details. (Applies only when the Printer Services is set to YES.)

Font Size — Font size to be used for printing, in points. (Applies only when the Printer Services is set to YES.)

The following table shows some sample entries.

Sample Print Spooler Information Table		
Field	**Sample Entry**	**Explanation**
File Separator Sequence	/0C	Is a command sequence consisting of control characters. The /0C control characters instruct printing device /dev/tty1 to insert a formfeed after each file it prints. This ensures that each file printed begins on a new page.
Status TAG	device1_status	Is an analog tag containing the status of the printing device. Other tasks can read the value of device1_status to determine whether the printer is busy before they send output to it.
Message TAG	device1_msg	Is a message tag containing a message about the status of the printing device. This message is displayed on a graphics screen by animating a Text object in Client Builder to display the value of this tag.

SAMPLE PRINT SPOOLER CONFIGURATION FOR ALARM LOGGER

Follow the steps below to configure Print Spooler for Alarm Logger:

1. Verify printer is configured in printer manager with a capture port assignment even if it is hooked up to LPT1.
2. Define printer in Spooler in one of the following ways:

```
LPT1
OS Services = NO
```

or

```
\\SERVICE\PRINTER_NAME OR PRINTER_NAME
OS Services = YES
```

or

```
\\SERVICE\PRINTER_NAME
OS Services = NO
```

3. Define printer device number = 1 in the alarm setup table.

ADDITIONAL INFORMATION FOR PRINTING TO LASER PRINTERS

In printing active alarms online using Windows OS services, you may get a form feed after each alarm. The reason this occurs is that every print job becomes a file and Windows adds the page break automatically. This is especially true with laser printers, like the HP LaserJet.

If the printer is connected directly to the computer, define the printer as generic text only, by using the Add Printer wizard to create a new printer and assign it to a open LPT port. When it asks for the manufacturer, select Generic and then select Generic / Text Only. Once the printer is set up, configure the Print Spooler. Under the Device column enter the LPT port you configured the printer on, like LPT1. Under the Use OS Print Services column, select NO. Then the page break will not be added between alarms. If you want to restore the printing of one alarm per page you can add a line feed (\0A) to the File Separator Sequence column in the spooler task.

If the printer is a network printer, then you have to map the network printer to an unused local printer port like lpt2. From a command prompt type:

net use lpt2: \\ComputerName\mylaserprintername yes

This maps the printer to lpt2, and the yes at the end sets the printer to be restored every time the user connects. In Print Spooler, set up lpt2 and set Use OS print services to NO. Then the printer will not print out a page until it is filled with alarms.

If the printer driver does not have the functionality to control whether or not a form feed is done, removing the printer from the print manager should make Spooler print directly to the port. There should not be a form feed, in this case.

Note that for page printers like HP laser jets, a whole page has to be filled or a form feed needs to be encountered before a page comes out of the printer. The form feed will be lighted but nothing will happen until enough alarms are generated to force it to start a second page.

PROGRAM ARGUMENTS

Argument	Description
-D<#>	Sets debug log level for Run-Time Manager output window. (# = 1 to 9)
-L	Enables logging of debug information to a log file.
-M	Use OS print services; send print requests (except for alarm logs and binary files) to the system print queue instead of directly to the printer.

ERROR MESSAGES

Error Message	Cause and Action
Bad CT file *filename*	Cause: The task cannot access information in the Print Spooler Information table. The specified table may be corrupt. Action: Delete **/FLAPP/CT/SPOOL.CT**. Restart to recreate the file.
Bad device number *number* in flags	Cause: An incorrect device is specified in a request sent to the Print Spooler. Action: Verify the entries in the Device field of the Print Spooler Information table. Check the task configuration tables for any task sending requests to the Print Spooler.
Can't open CT file *filename*	Cause: Either the **/FLAPP/CT/SPOOL.CT** file does not exist or is corrupt. Action: Delete the file if it exists and restart to recreate the file.
Can't open input file *filename*	Cause: The input file may not exist or may be opened by another task. Action: Check that the specified file exists.
Can't open output device *device_name*	Cause: If the data is being written to a file, an incorrect path may be specified. If the output device is a printer, the printer may be busy. Action: Ensure the specified path is correct if the output device is a file. Check the specified output device (such as a printer) for problems.
Hard error on device *device_name*	Cause: A problem exists with the specified output device. Action: Check the specified output device for problems, such as if the printer is offline.
Illegal flags flag for output device *device_name*	Cause: An incorrect option flag is specified in a request sent to the Print Spooler. Action: Contact your technical support representative.
Illegal printer sequence for device *device_name*	Cause: An incorrect sequence is entered in the Print Spooler Information table for the specified device. Action: Verify that the entries for File Begin, File End, Binary On, Binary Off are correct for the type of device specified in the Device field.

Error Message	Cause and Action
I/O error writing to device *device_name*	Cause: The **FLAPP/CT/SPOOL.CT** file may be damaged or you may have a problem with the output device. Action: Ensure the output device, such as a printer, is functional. When it is, delete **FLAPP/CT/SPOOL.CT** and restart the application to recreate the file.
No colon in flags	Cause: A colon was not included in a request sent to the Print Spooler. Action: Contact your technical support representative.
No output devices	Cause: No output device is specified in the Print Spooler Information table. Action: Specify a device in the Print Spooler Information table. Rerun the application.
Not enough RAM **Not enough RAM to continue task**	Cause: Not enough RAM is available to perform this task. Action: Close any unnecessary windows or programs, such as the Client Builder or any text editor. Add RAM to the system if this error message occurs often.
Spool: Task initialization failed	Cause: Either Print Spooler is already running or is not enabled for your configuration sequence. Action: Check that the Print Spool option is enabled.
Spooler backup file too large. New data not saved.	Cause: The printer is out of paper or offline. Alarm logging data has been saved to a temporary file. This file holds up to 640,000 bytes of data, and the file has now exceeded this size. Any new alarm logging data is not saved. Action: Put the printer back online or put paper in the printer. When the printer is ready, data saved in the temporary file is spooled to the printer, and the data in the temporary file is deleted. Alarm logging data is spooled to the printer.
Unknown flag *flag* for output device *device name*	Cause: An illegal entry is made in the Print Spooler Information table for the specified output device. Action: Verify the entries in the Print Spooler Information table.

Chapter 19. . . .

Programmable Counters

The Programmable Counters task provides totalizers and event delays, such as defining a trigger to unlock a door and then specifying a delay before the door locks again. A programmable counter is similar to a counter in programmable controllers.

Operating Principles

A programmable counter is a group of tags with values that work together to perform a count. Outputs from programmable counters can be used to provide input to or trigger Math & Logic programs or other FactoryLink tasks. There is no limit, except the amount of memory, to the number of programmable counters that can be defined.

Each programmable counter is made up of some or all of the following tags and analog and digital values.

Tags

- Enable—triggers counting activity.
- Up Clock—initiates the count upward.
- Down Clock— initiates the count downward.
- Clear—resets the counted value to the starting point.
- Positive Output—contains the value 1 (on) when the counting limit has been reached.
- Negative Output—contains the value 0 (off) when the counting limit has been reached.
- Current Value—indicates the current value of the count.

Digital and Analog Values

- Preset Value—analog value that specifies the starting value.
- Increment Value—analog value that specifies the amount by which the count is to increase or decrease each time.
- Terminal Value—analog value that specifies the counting limit.
- AutoClear—digital value that resets the count to the starting point whenever the terminal value is reached.

Counting begins when another FactoryLink task, such as Math & Logic or EDI, writes a 1 (ON) to the Up Clock tag. This triggers the Programmable Counters task to move the Current Value toward the Terminal Value by the Increment Value. If the Preset Value is less than the Terminal Value, the Increment is added to the Current Value. If the Preset Value is more than the Terminal Value, the Increment is subtracted from the Current Value.

There is no limit, except the amount of memory, to the number of programmable counters that can be defined.

Example One

In this example, counting is configured to count bottles (20 per case). The Preset Value (start count) is 0 and the Terminal Value (count limit) for the number of bottles per case is 20. The Increment Value of 1 represents one bottle. When counting is triggered, each bottle counted increases the current count of bottles (starting with 0 in the case) by 1 until the case contains 20 bottles (until the Current Value reaches the Terminal Value of 20).

When the case contains 20 bottles (when the Current Value reaches the Terminal Value), the Counter task indicates the case is full by force-writing a 1 (ON) to the Positive Output tag and force-writing a 0 (OFF) to the Negative Output tag. At this point, if AutoClear = YES, the Current Value tag is reset to 0 (the Preset Value) and the count can begin again. If AutoClear = NO, the current Value tag remains at 20 (the Terminal Value) until another task writes a 1 (ON) to the clear tag, indicating the count can begin again. The count does not continue past 20 (the Terminal Value). Each time the bottle count reaches 20 (the Terminal Value), the Counter task again force-writes a 1 (ON) and a 0 (OFF) to the Positive and Negative Output tags. When AutoClear = YES or when the Clear tag is triggered, the bottle count is reset to 0 (the Preset Value), ready for a repeat of the counting process.

Example Two

You can set up another task, such as EDI or Math & Logic, to react to a deviation, such as a defective bottle, during the count by adjusting the count. To adjust the count, that task writes a 1 (ON) to the Down Clock tag to cause the value of the Current Value tag to move toward the Preset Value by the Increment Value.

For example, during counting, if a defective bottle is counted but not packed in the case, the EDI or Math & Logic task subtracts that bottle from the total count by writing a 1 (ON) to the Down Clock tag to cause the Current Value to move toward the Preset Value (0 in this example) by the Increment Value (1 in this example).

After six bottles have been counted and packed in the case, the Counter task counts the seventh bottle. But the seventh bottle is defective, so it is not packed in the case. Therefore, the EDI or Math & Logic task subtracts that bottle from the total count by writing a 1 (ON) to the Down Clock tag. This causes the Current Value to move from 7 down to 6.

PROGRAMMABLE COUNTERS INFORMATION TABLE

The Programmable Counters task uses either Shared or User domain. The Shared domain is recommended unless the counters need to be unique to each user for the purposes of the application.

Accessing

In your server application, open **Timers > Programmable Counters > Programmable Counters Information.**

Field Descriptions

Enable
: Tag that enables or triggers counting. If this field is left blank, counting is always enabled because the trigger becomes either the UP CLOCK or the DOWN CLOCK. When the value of Enable is set to 1 (ON), counting occurs. If the value of Enable is set to 0 (OFF), counting does not occur.

  Valid Entry: tag name
  Valid Data Type: digital

Up Clock
: Causes the value of the Current Value tag (present count) to move toward the Terminal Value (count limit). When a 1 (ON) is written to the Up Clock tag, the value in the Current Value tag is increased by the amount specified by the Increment Value tag. If the Preset Value (starting count) is less than the Terminal Value, the Increment Value is added to the Current Value. If the Preset Value is greater than the Terminal Value, the Increment Value is subtracted from the Current Value.

  An entry is required in either the **Up Clock** or **Down Clock** field.

  Valid Entry: tag name
  Valid Data Type: digital, analog, floating point, longana, message

Down Clock
: Causes the value of the Current Value tag (present value) to move away from the Terminal Value (toward the Preset Value). When a 1 (ON) is written to the **Down Clock** tag, the value in the Current Value tag is decreased by the amount specified by the Increment Value tag. If the Preset Value is less than the Terminal Value, the Increment Value is subtracted from the Current Value. If the Preset Value is greater than the Terminal Value, the Increment is added to the Current Value. The Current Value does not move past the Preset Value, and the Positive/Negative Outputs are not triggered when the Preset Value is reached.

An entry is required in either the **Up Clock** or **Down Clock** field.

Valid Entry: tag name
Valid Data Type: digital, analog, floating point, longana, message

Clear

Causes the Current Value to be reset to the Preset Value each time a 1 (ON) is written to it (the Clear tag). Each time a 1 (ON) is force-written to the Clear tag, a 0 (OFF) is force-written to the Positive Output tag and a 1 (ON) is written to the Negative Output tag.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message

Positive Output

Name of a tag a 1 (ON) is written to each time the Current Value reaches the Terminal Value. The value of the Positive Output tag remains 1 (ON) until a 1 (ON) is written to the Clear tag.

Valid Entry: tag name
Valid Data Type: digital

Negative Output

Name of a tag a 0 (OFF) is written to each time the Current Value reaches the Terminal Value. The value of the Negative Output tag remains 0 (OFF) until a 1 (ON) is written to the Clear tag. This tag is set to 1 at task startup.

Valid Entry: tag name
Valid Data Type: digital

Current Value

Contains the current value of the counter. This value is always between the Preset Value and the Terminal Value.
The default is 0.

Valid Entry: tag name
Valid Data Type: analog, longana

Preset Value

The starting value for a count. This numerical value is written to the Current Value tag whenever the value of the Clear tag is set to 1 (ON).

Valid Entry: -32768 to 32767
Default: 0

Increment Value

The numerical value that is combined with the Current Value when the **Up** or **Down Clock** is triggered.

Valid Entry: 0 to 32767
Default: 1

Terminal Value — Specifies a limit for counting activity. When the Current Value is the same as the Terminal Value, counting stops and Positive and Negative Outputs are triggered. Counting remains stopped until the Clear tag is triggered; however, if AutoClear is set, a Clear is performed immediately after the Positive and Negative Outputs are triggered.

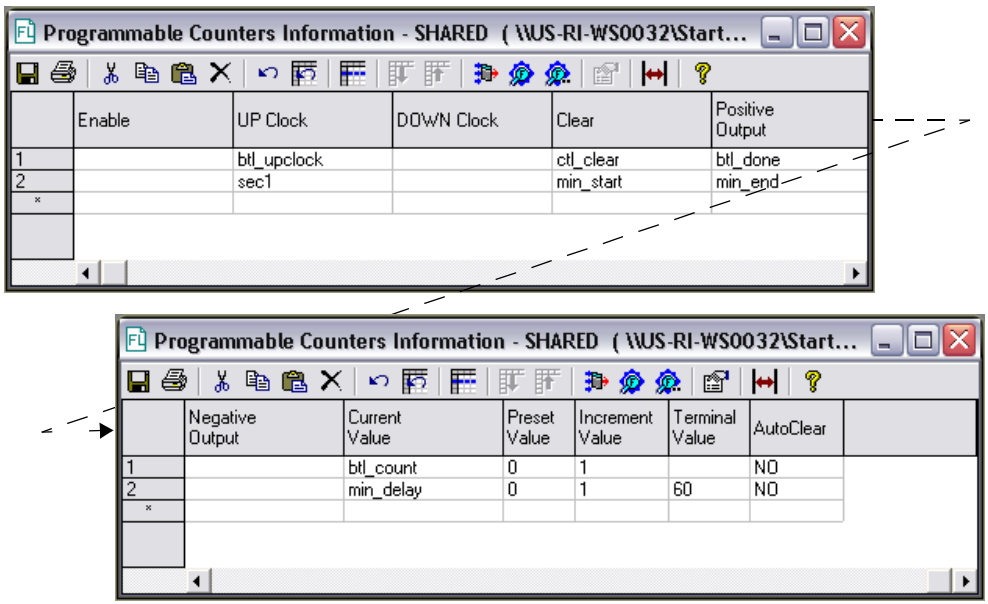
Valid Entry: -32768 to 32767

Autoclear — Indicates whether to automatically clear the counter each time the Terminal Value is reached.

YES — Clear is performed each time the Terminal Value is reached. (default)

NO — Current Value remains equal to the Terminal Value until Clear is triggered.

Example



The counter in the first line on the table, along with a Math & Logic procedure that saves the count and resets the counter, counts the number of bottles packed per minute. Since the **Enable** field is left blank, counting is always enabled. Each time a bottle is packed, a 1 (ON) is written to the **Up Clock** tag **btl_upclock**. This triggers the Counters task to increase the Current Value **btl_count** by 1. Each minute, FactoryLink triggers a Math & Logic procedure to log the Current Value and trigger the Clear tag **btl_clear** to reset the count for the next minute.

In the second line on the table, the counter is used to create a one-minute delay of an event, such as bottle capping. Since the **Enable** field is left blank, counting is always enabled. When the value of **sec1** becomes 1 (ON), the Counters task increases the Current Value **min_delay** by 1. The task continues to increase this value once each second until the Current Value matches the Terminal Value of 60. At this time, counting stops and the Counters task writes a 1 (ON) to the Positive Output tag **min_end**, indicating the end of the one-minute delay. Other FactoryLink tasks can monitor the **min_end** tag to trigger another operation and then write a 1 (ON) to the Clear tag **min_start** to reset the count.

PROGRAM ARGUMENTS

Argument	Description
`-t`	The Programmable Counters task establishes parameters for the initiation, performance, and conclusion of counting activity. With the **-t** program argument in the System Configuration for the Counters task, negative output, positive output, and current value are initialized. Positive output is set to 0. Negative output is set to 1. With no program argument, those tags remain at their default/persistent values.

ERROR MESSAGES

Error Message	Cause and Action
Can't start process 'COUNTER'	Cause: The default number of processes may be less than FactoryLink requires to start up because the operating system parameters were not set properly during FactoryLink installation and setup. Action: Contact your technical support representative.
Corrupt configuration table index **Corrupt data in information panel** **Invalid CT record size**	Cause: The **/FLINK/CT/CNT.CT** file is damaged. Action: Delete the file. Restart the application to rebuild the file.
No counter tag defined	Cause: You did not define an Up Clock, Down Clock, or Clear tag. Action: Define at least one of these tags.
No counters defined	Cause: You have not defined any counters for the Programmable Counters task. Action: Define some counters or remove the **R** flag for the task from System Configuration Information table.
No tables configured for this task	Cause: **FLINK/CT/CNT.CT** does not exist or cannot be opened. The installation may not have completed successfully or the file may have been damaged. Action: Delete the file. Restart the application to rebuild the file.
Out of RAM	Cause: No more memory is available. Action: Close any unnecessary windows or programs, such as the Client Builder or any text editor. Add memory to the system if this error message occurs often.
Value tag must be analog	Cause: The data type of the Current Value tag is invalid. Action: Ensure the Current Value tag is an analog tag.

Chapter 20. . . .
Report Generator

DEFINING THE REPORT FORMAT

Data FactoryLink collects or computes is stored as tags in the real-time database. Each time data is collected or computed, the value stored in the real-time database for a tag is overwritten by the new data.

If you want to report on this real-time data, you can write the data to a report file as it is received using Report Generator. The Report Generator is a flexible reporting tool that lets you define simple custom reports. The data included on the report can be generated as a disk file, a printed report, or exchanged with other programs that accept ASCII files.

Some typical uses for generating report data include the following:

- Predicting potential problems based on data patterns
- Reporting on productivity of shifts
- Generating hardcopy reports for management or specific agencies

> **Note:** Depending upon the types of reports you need, you might also want to consider using the predefined Historical Reports that are available with FactoryLink or use a reporting tool to create reports from data stored in a relational database.

Reporting Methodology

Memory-resident real-time data is logged to a report file for generating a report. This task completes the following steps to generate a report:

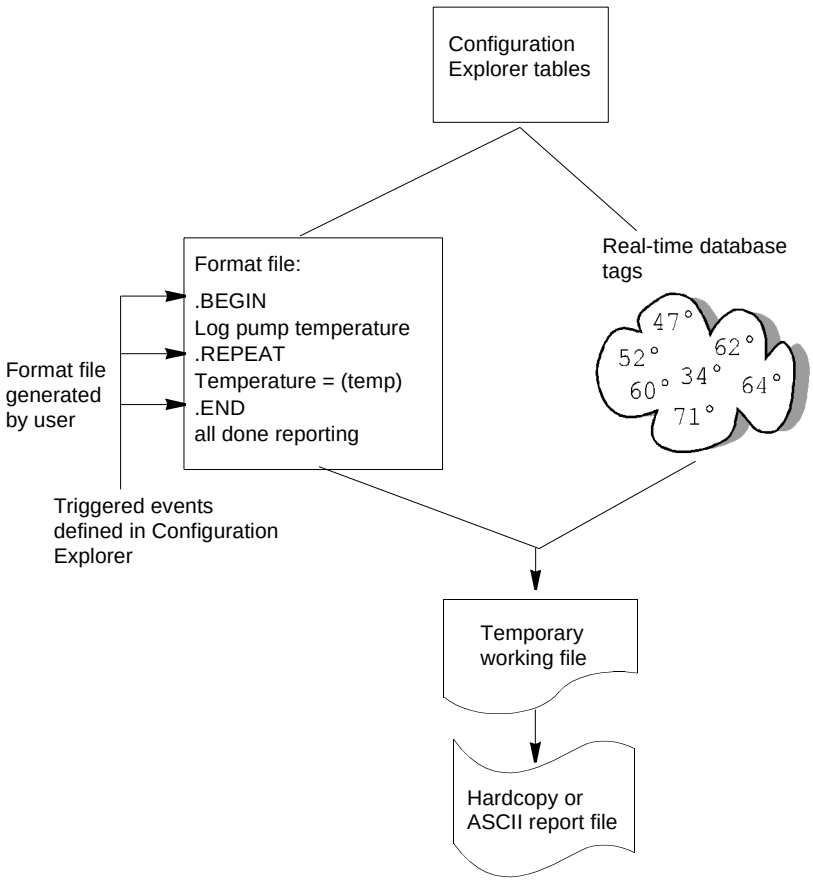
1. The real-time database receives data from various sources, such as a remote device, user input, or computation results from a FactoryLink task.
2. When a report is triggered, Report Generator reads the current values of the tags included on the report and maps them to object names. Object names are used in defining the report format or template file.
3. Report Generator checks the report format file to determine placement of text and objects in the report file. The format file contains keywords that trigger when the report starts, ends, and writes data. Each keyword represents a section. When the trigger executes, the associated section of the format file is processed and written to a temporary working file.

4 Report Generator uses the information in the report format file to create a temporary disk-based working file. This working file is a temporary file that remains open until the report completes and exists only until the report is completed.

The temporary file resides on disk, not in memory, to protect against loss of data. For example, if FactoryLink shuts down before Report Generator has created the report archive file, the temporary report file still exists on disk.

5 When the report is completed, the information in the temporary disk-based working file is sent to either a permanent file on disk, a printer, or a communication port.

The following illustration outlines the process of generating a report.



Format File Components

User-defined ASCII format files control the look and contents of the report. A unique format file is defined for every report generated by Report Generator. This section describes the components of the format file.

Keywords

Keywords are used in the format file to trigger an action. The associated section of the format file is processed and written to a temporary working file when the trigger executes. Three keywords are used in format files:

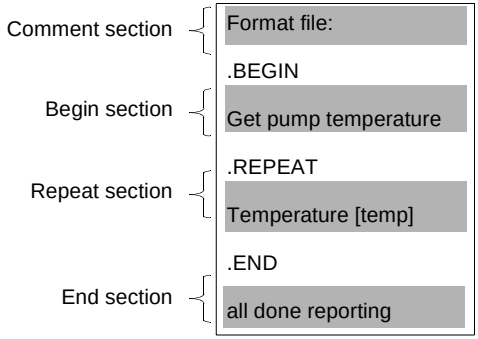
- .BEGIN
- .REPEAT
- .END

Keyword lines begin with a period (.), followed by a keyword and a line terminator such as LF (Line Feed) or CR, LF (Carriage Return, Line Feed) sequence.

Comment lines can also be included in the format file by starting the line with a period and following it with any text that does not represent a keyword. Text displayed in a comment line is not included in the report.

Format File Sections

The report format file has four distinct sections:



The sections and their associated keywords are:

- Comment section—First section in any format file. Any text displayed in this section is not included on the report. This section usually includes a description of the report purpose. Do not precede the text in this section with a period.
- Begin section—Delineated by .BEGIN and defines the report header. Any text displayed after the .BEGIN keyword and before the next keyword is included at the beginning of the report.
- Repeat section—Delineated by .REPEAT and contains the object names corresponding to tags to include in the report. Each time the repeat section executes, tag values are written to the temporary working file. This section can also include literal text included in the report.
- End section—Delineated by .END. This section ends the report and contains text to include at the end of the report.

> **Note:** If reports are stopped and started before the end trigger, when the task is restarted and the begin (or repeat if no begin trigger) is activated, the system will check to see if a report file already exists. If it does, the system will not add another header.

Each format file consists of one or more of these sections. The Begin, Repeat, and End sections can include object names that are substituted with tag values when the report is generated. The only required section is the End section, which generates a snapshot report if used alone.

Data specified in the format file is collected from the real-time database and placed into the report. Placement of real-time database values is determined by the following:

- Location of its object name in the format file
- Format specifiers

Object Name Location

Object names act as a placeholder for data and are linked to tags in the real-time database. The value of the tag replaces the object name during report generation. Object names are enclosed in braces {} or brackets [ ] within the format file.

- Use braces { } for data that may vary in length. This places the data relative to other text in that line because the position may change based on the tag value. A typical use may be to locate data within a sentence.
- Use brackets [ ] for fixed position data. The value of the tag associated with the object name is displayed in the report exactly where the object name is displayed. The starting bracket, which is the anchor for the data, is typically used to format data in columns.

The identifier (braces or brackets) is not displayed in the generated report file. Use an escape sequence identified in “Escape Sequences” on page 470 if you want a brace or bracket to be displayed in the report.

You can use object names in the begin, repeat, and end sections.

Format Specifiers

Format specifiers allow you to define a variable where a literal is expected. Format specifiers consist of two types of objects:

- Ordinary characters, which are copied literally to the output stream
- Variable specifiers, which indicate the format variable information are displayed in

Format specifiers use the following form: *% [flags][width][.prec]type*

The following table provides a list of the specifiers typically used with Report Generator.

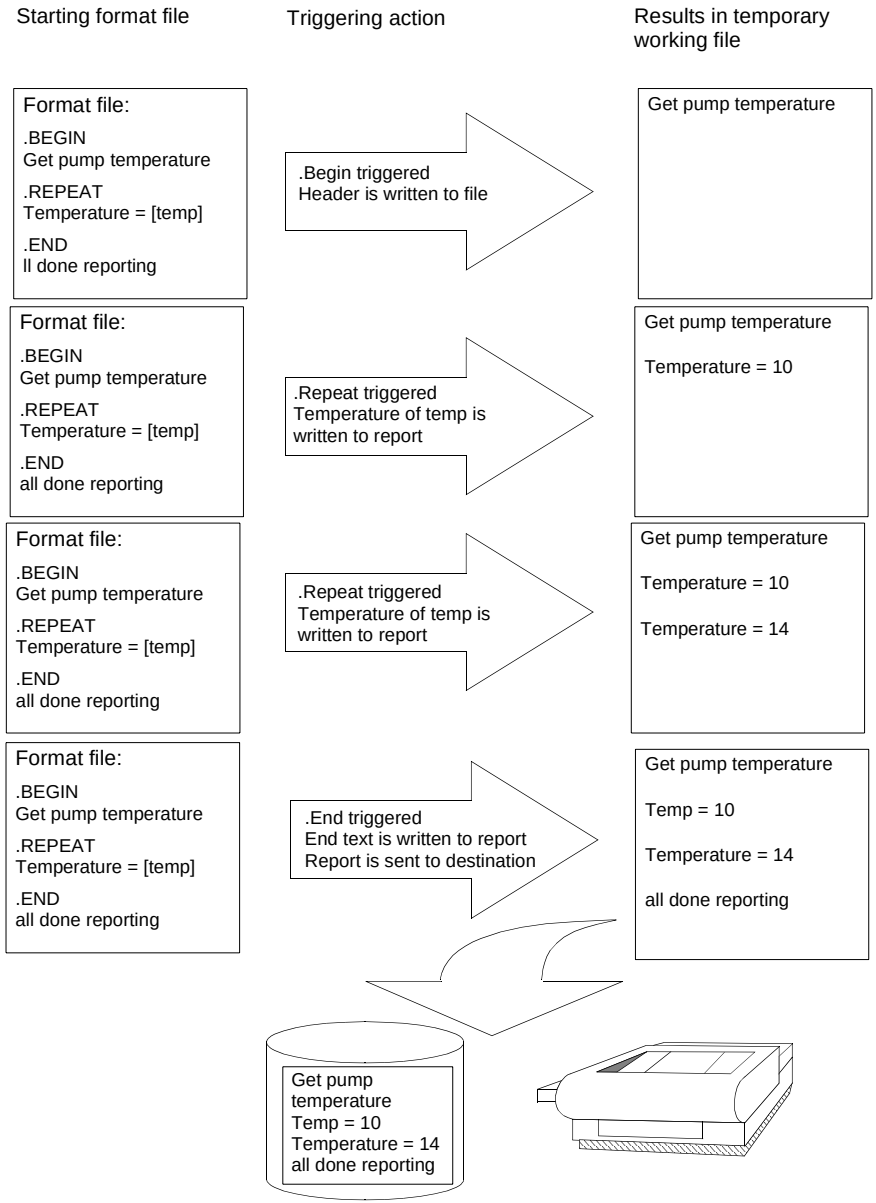
Sample Archive File	Report Archive File Tag	Value of Archive File Tag	Actual Archive File Path Generated
C:\USCO\REPORT.%03d	A_DOY	33	C:\USCO\REPORT.033
C:\USCO\%s.RPT	shift	“FIRST”	C:\USCO\FIRST.RPT
C:\USCO\PUMP%d\RPT.RPT	pump_no	1	C:\USCO\PUMP1\RPT.RPT

Trigger Actions

When a .BEGIN, .REPEAT, or .END trigger executes, the associated section of the format file is processed and written to a temporary working file.

The following figure illustrates what occurs when each keyword is triggered. This sample report format is used to generate an historical data log. A temporary working file is opened when the report is triggered. This file remains open until the end section is triggered. The report header is written to the file when the begin section is triggered. In this example, **Get pump temperature** is written at the top of the report.

When the repeat section is triggered, the values of the tags mapped to the object names included in this section are read from the real-time database and written to the file. In this example, the value of the tag containing the pump temperature is mapped to the object name **temp** and is written to the report.


Starting format file
Triggering action
Results in temporary working file
Format file:
.BEGIN
Get pump temperature
.REPEAT
Temperature = [temp]
.END
ll done reporting
.Begin triggered
Header is written to file
Get pump temperature
Format file:
.BEGIN
Get pump temperature
.REPEAT
Temperature = [temp]
.END
all done reporting
.Repeat triggered
Temperature of temp is written to report
Get pump temperature
Temperature = 10
Format file:
.BEGIN
Get pump temperature
.REPEAT
Temperature = [temp]
.END
all done reporting
.Repeat triggered
Temperature of temp is written to report
Get pump temperature
Temperature = 10
Temperature = 14
Format file:
.BEGIN
Get pump temperature
.REPEAT
Temperature = [temp]
.END
all done reporting
.End triggered
End text is written to report
Report is sent to destination
Get pump temperature
Temp = 10
Temperature = 14
all done reporting
Get pump temperature
Temp = 10
Temperature = 14
all done reporting

Any literal text included in this section is also written to the file. In this example, the literal text **Temperature =** is written to the report in front of the tag value. The event that triggers the repeat can be a periodic sampling, a specific time, or an event driven trigger like a part meeting a photo-eye in a conveyor system.

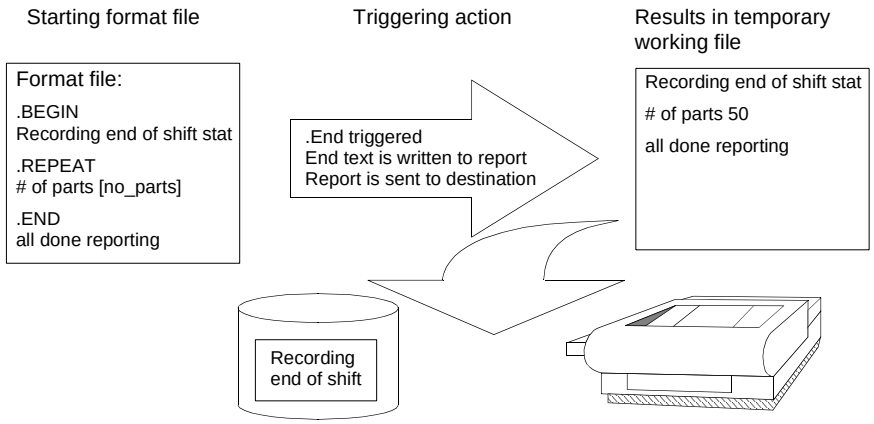
You can trigger the repeat section any number of times before ending the report. In this example, the pump temperature is written to the report twice. The first time its value is 10; the next time its value is 14.

The literal text in this section is written to the temporary working file when the end section is triggered; then, the entire report is sent to its configured destination. This can be a disk-based file, a printer, or across the network to another node. The temporary working file is deleted.

Report Format Variations

A simple variation on the report used in the previous example can be generated by not specifying an event or trigger for the begin section. This is typically used if no header type information is necessary. An end section must always be specified.

Another common format for reports is a snapshot report, as shown in the following figure. The purpose of this type of report is to gather information and to generate a printed report by triggering a single event. This is done by specifying only an end trigger. The end event causes all information in the format file to be sent instantly to the printer.



Complete Triggers

FactoryLink returns a complete status for each of the triggered events once the operation has completed. The operation is considered complete for begin and repeat sections when the information is written into the temporary file. The operation is considered complete for the end section when the temporary file is sent to the specified device (printer, communication port, or disk file). The complete status allows you to effectively coordinate report generator operations with other FactoryLink tasks and to display the status to the operator.

If the data reported on in the repeat section is generated from an external device connected via a device interface task, in order to maintain data integrity, it may be necessary to coordinate operations between these two tasks. You do not want to log data to the report unless you are certain the data has been returned successfully from the device interface task. Likewise, you do not want to sample more data from the external device before the previous data is logged through the report generator.

Another application that may require coordination is if you want to read data from a relational database using Browser and write it to the printer. You can do this by using the complete trigger on the Browser to trigger the Report Generator repeat trigger and then have the Report Generator complete trigger generate a move to the next database row in the Browser task. This coordination takes place until all rows are fetched. Use Math and Logic to verify not only complete, but successful completion, with both tasks.

Escape Sequences

Escape sequences send instructions to the printer, such as form and line feeds. These sequences can also be used to change operating modes of printers to compressed versus standard print. The following table lists and explains commonly used escape sequences.

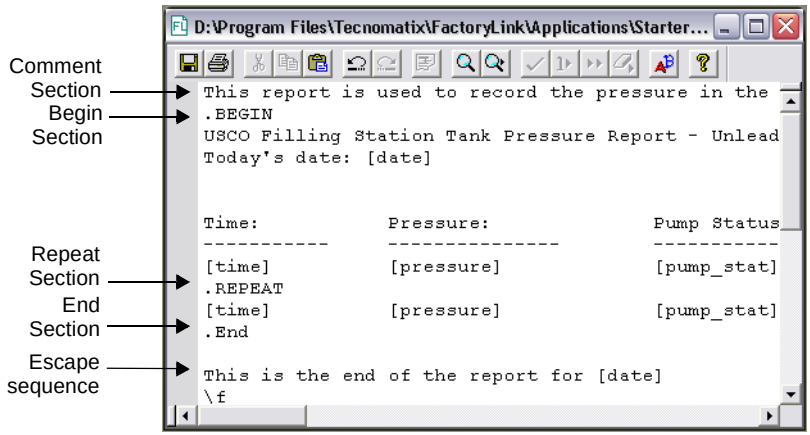
Escape Sequence	Description
\b	Send backspace (0x08).
\f	Send form feed (0x0C).
\n	Send line feed (0x0A).
\r	Send carriage return (0x0A).
\t	Send horizontal tab (0x09).
\XX	Send 0xXX or any two uppercase hex digits (\9F).
\Z	Send Z, where Z is any character not previously listed.
\.	Send. (necessary to start a Report File line with a period.)
\[	Send [.
\{	Send {.
\\	Send a single \.

CONFIGURING REPORT FORMAT FILES

A format file is required to determine the placement of the generated data when a reporting operation is triggered for Report Generator. This format file is created in the Report Generator Format table, a free-form text entry table.

You must define a unique format file for every report. Format files are stored by default in the **FLAPP/rpt** directory as *filename*.fmt where *filename* is the name you assign to the format file.

Figure 20-1 Sample Report Format File



Perform the following steps to define a format file:

1 In your server application, navigate to **Reports > Report Generator Formats**.

2 Right-click **Report Generator Formats** and select **New Report Format File**.

3 Type a unique report format file name and click **Enter**. FactoryLink automatically adds the **.fmt** extension.

4 (Optional) Enter a comment in the comment section starting with the first line of the format file table. The comment section extends to the first line starting with a keyword. Each line in the comment section cannot exceed 512 characters. It is not necessary to precede comments in this section with a period. A comment can reference the format file or the report you are configuring. In Figure 20-1, the sample report contains one line of comments.

5 (Optional) Define a begin section by entering the keyword .BEGIN followed by text you want as the header for the report. Enter the name of the report, column names, and any other fixed data in this section. You can also include object names, such as date and time. It is not required to include a begin section.

In Figure 20-1, the Begin section includes the report name, the report date, column names, and initial data values. The date and tags identified by object names are read from the real-time database and inserted into a fixed position on the report denoted by the brackets.

6 (Optional) Define a repeat section by entering the keyword .REPEAT followed by any text and names of objects you want included in the report. You can include both text and object names in this section. The contents of this section can be repeated in the report any number of times. This section is repeated each time the repeat trigger is activated.

In Figure 20-1, the Repeat section includes data to be read from tags and inserted into a fixed location in the report. The Repeat section is completed when the last object name is read and sent to the temporary file at the end of the first shift.

7 Define an End section by entering the keyword .END followed by text and object names you want included at the end of the report.

At a minimum, a report format file must include an end section. In Figure 20-1, the example format file has one line End section that includes literal text and an object name that places the date the report is completed.

8 (Optional) Enter an escape sequence to specify instructions to the printer. If you do not enter an escape sequence, the report prints exactly as defined in this format file. In Figure 20-1, the example contains an escape sequence instructing the printer to form feed the paper when printing completes.

9 After you finish formatting the report, save the file and then close it.

CONFIGURING REPORT TABLES

Reporting operations are defined in the Report Generator configuration tables:

- Report Generator Control—Defines the parameters of the report, such as the name of the temporary file, archive file name, report output device, and conditions that trigger the report.
- Report Generator Information—Defines the object names used in the format file and links them to tags in the real-time database.

Report Generator Control Table

The parameters of the report, such as the name of the temporary file, archive file name, report output device, and conditions that trigger the report are defined on the Report Generator Control table. Specify information in this table for each report you want generated. Each report is a line entry in the pane.

Accessing

In your server application, open **Reports > Report Generator > Report Generator Control**.

Field Descriptions

Report Name
: Name of the report. This name must match the name entered in the **File Name** field of the report format file. Do not include the **.fmt** extension.

  Valid Entry: report format file name (without the .fmt extension)

Report Temporary File
: Leave this field blank unless it is necessary to recover temporary report files in the event of abnormal termination. Report Generator uses the name of the report defined in the **Report Name** field with an **.rpt** extension as the temporary file name. This file is placed in the following directory path:

  {FLAPP}/FLNAME/FLDOMAIN/FLUSER/rpt

  Enter the name to assign to the temporary working file created when the report is started if you need to recover temporary working files. Include this name in the directory path where you want the temporary file opened. This path cannot exceed 32 characters. The directory path must exist with proper permissions prior to running the report.

  Use variable format specifiers as part of the path and file name, so you can create new temporary files without overwriting previously created temporary files. The format specifier included in the definition is replaced by the value defined in the **Report Temporary File Tag** field when the report is triggered and the temporary file is created. You must define the **Report Temporary File Tag** field if you specify a variable specifier as part of the path or file name.

Report Temporary File Tag
: Contains the value to use to replace the variable specifier defined in the **Report Temporary File** field. The variable specifier is interpreted as a literal if you leave this field blank and you used a variable specifier when you defined the temporary file name. Make the variable specifiers the same type as their corresponding tags.

  Valid Entry: tag name

  Valid Data Type: digital, analog, longana, float, message

Report Archive File
: Name to give the permanent report file created when the report completes. The contents of the temporary working file are written to this file.

  If you leave this field blank, Report Generator does not save the report and deletes the temporary file when the reporting operation completes. The report can no longer be accessed.

This name can include the directory path where you want the file created. The directory path must exist with proper permissions prior to running the report. If you do not define a directory path, the report is created in the following directory path:

{FLAPP}/FLNAME/FLDOMAIN/FLUSER/rpt

You can use variable specifiers as part of the path and file name to generate reports without overwriting previously generated reports.

The format specifier included in the definition is replaced by the value defined in the **Report Archive File Tag** field when the report completes and the permanent file is created. You must define the **Report Archive File Tag** field if you specify a variable specifier as part of the path or file name.

The file is overwritten the next time the reporting operation completes and the temporary file is written to this archive file if you configure the report archive file name without variable specifiers.

Valid Entry: permanent report file name

Report Archive File Tag

Contains the value to use to replace the variable specifier defined in the **Report Archive File** field.

The variable specifier is interpreted as a literal if you leave this field blank and you used a variable specifier when you defined the archive file name. Make the variable specifiers the same type as their corresponding tags.

Valid Entry: tag name
Valid Data Type: analog

Begin Trigger

Triggers the execution of the begin section of the format file. The begin section is triggered by the end trigger if you leave this field blank and the format file contains a begin section.

When this tag value is forced to 1 (ON), data defined in the begin section of the format file is written to the temporary file defined in the **Report Temporary File** field.

Valid Entry: tag name
Valid Data Type: digital

Begin Complete

Contains the status of the begin operation. Report Generator force-writes this tag to 1 when the begin section of the report completes. Execution of the begin section is considered complete when the contents of the begin section are written to the temporary file.

Leave this field blank if your report does not include a begin section or if you do not want to maintain the status of the begin operation.

Valid Entry: tag name
Valid Data Type: digital

Repeat Trigger

Tag that triggers the execution of the repeat section of the format file. Leave this field blank if your report does not include a repeat section. The repeat section is triggered by the end trigger if you leave this field blank and the format file contains a repeat section.

When this tag value is forced to 1 (ON), data defined in the repeat section of the format file is written to the temporary file defined in the **Report Temporary File** field.

Valid Entry: tag name
Valid Data Type: digital

Repeat Complete

Tag that contains the status of the repeat operation. Report Generator force-writes this tag to 1 when the repeat section of the report completes.

Execution of the repeat section is considered complete when the contents of the repeat section are written to the temporary file.Leave this field blank if your report does not include a repeat section or if you do not want to maintain the status of the repeat operation.

Valid Entry: tag name
Valid Data Type: digital

End Trigger

Tag that triggers the execution of the end section of the format file. Do not leave this field blank. When this tag is forced to 1 (ON), data defined in the end section of the format file is written to the temporary file defined in the **Report Temporary File** field.

If the format file contains begin and repeat sections but begin and repeat triggers are not defined, the end trigger executes these sections before executing the end section. Once the end section is executed, the contents of the temporary file are sent to its configured destination and the temporary file is deleted.

Valid Entry: tag name
Valid Data Type: digital

End Complete

Tag that contains the status of the end operation. Report Generator force-writes this tag to 1 when the repeat section of the report completes.

Execution of the end section is considered complete when the contents of the end section are written to the temporary file and the temporary file is sent to the specified device (printer, communication port, or disk file).

Valid Entry: tag name
Valid Data Type: digital

Printer Device Enter a number between 1 and 5 that corresponds to the printer defined in the Print Spooler table where you want to print the report. You can direct reports to any of five different spooler devices specified in the Print Spooler Information table.

Valid Entry: 1 to 5
0 or blank = (do not want to print the report)

Report Generator Information Table

The object names used in the format file and their association with tags in the real-time database are defined on the Report Generator Information table.

Accessing

In your server application, open **Reports > Report Generator > Report Generator Control > "your report name" > Report Generator Information.**

Field Descriptions

Add an entry for each object you defined in the format file. You can configure up to 2,048 entries in this table. Each row in this table represents an entry. If you do not have any object names defined in the format file, define a placeholder record using any valid FactoryLink tag and any object name as the placeholder. You are limited to 256 characters when formatting a line in the report.

Tag Name Tag you want included in the report. The value of this tag is written to the report in place of the object name holder defined in the format file. Tag names can be the same as the object names or they can be different.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float, message
Default: message

Object Name Name of the object exactly as defined in the report format file.

Format Variable specifier to indicate how to format the data in the **Tag Name** field when written to the report. The tag is displayed as it exists in the real-time database if you leave this field blank.

Valid Entry: specifier of up to 12 characters

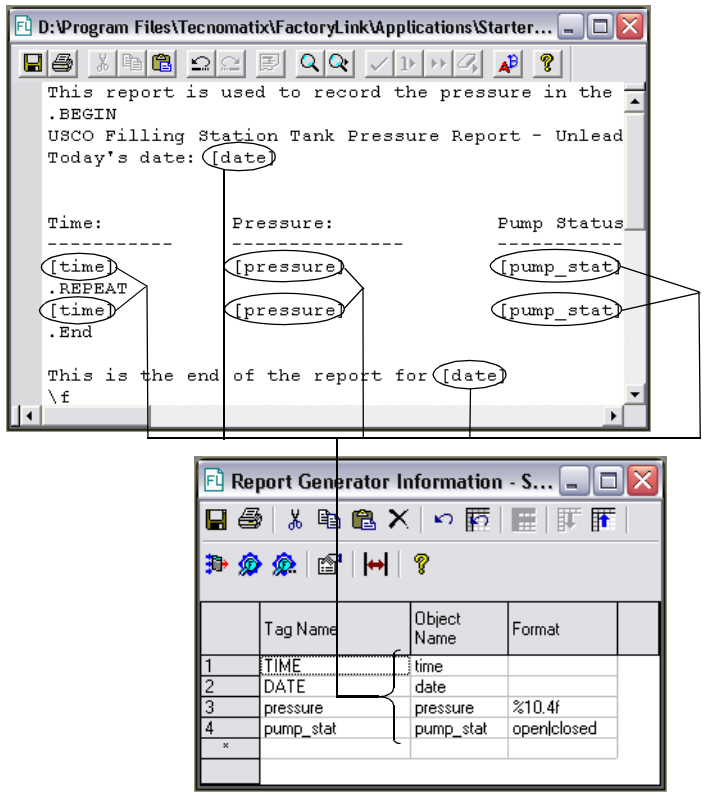
If the tag is digital, you can specify a character string to be displayed depending on the digital value. To do this, enter the desired character strings for 0 and 1 in the following format:

open|closed

where:

open	specifies the message **open** to print when the tag is 1
closed	specifies the message **closed** to print when the tag is 0

Example



In the example, the date and time objects do not have associated formats specified, so the data displays as defined in the tag definition. The object pressure is formatted to 10 total characters with 4 significant digits after the decimal point. The object **pump_stat** indicates the current status of the pump draining the tank (open or closed).

ERROR MESSAGES

Error Message	Cause and Action
Archiving report *report_name*	Cause: A status message indicates *report_name* is in the process of being archived Action: None required.
Bad header in CT file *report_name*	Cause: The CT entry associated with *report_name* is corrupt. Action: Run **ctgen -v2 rpt**. Note any errors and make corrections.
Bad prt device #%d in rpt %d	Cause: The device number %d is not a valid spool device. Action: Ensure the device number specified has a corresponding entry in the Spooler task table.
Bad type in CT file *report_name*	Cause: An unsupported or mismatched tag type is specified in the *report_name* entry in the Report Generator table. Action: Verify table entries. Force a regeneration of the CT file to ensure its integrity by running **ctgen -v2 rpt**. Note any errors and make corrections.
Can't open CT file *report_name*	Cause: The CT entry associated with *report_name* is either corrupt or nonexistent. Action: Verify Report Generator tables were configured in the Shared domain. Regenerate the CT file by running **ctgen -v2 rpt** and note any errors or corrections to be made.
Can't open input file *format_file*	Cause: The *format_file* specified in the configuration tables in the Report Generator Format file entry and the *report_name* do match or the *format_file* for the specified *report_name* does not exist. Action: Ensure a one to one correspondence between the *format_file* names and the *report_name* entries. Verify spelling and syntax.
Can't open output file *archive_file*	Cause: The file specified in the *archive_file* can not be created. Action: The path specified in the *archive_file* column must be created prior to running the Report Generator task. Ensure a valid path is resolved in a format specifier is used in the Archive file entry.

Error Message	Cause and Action
Can't spool output file *report_name*	Cause: The Report Generator is unable to send the report generated by the *report_name* entry to the Spooler task. Action: Ensure the spooler device specified in the Report Generator exists and the Spooler task is running with no errors.
Error reading configuration table record	Cause: Unable to read information in the Report Generator Information table. Action: Run **ctgen -v2 rpt**. Note any errors and make corrections if necessary.
Error writing file *archive_file*	Cause: The file specified in the *archive_file* can not be created or closed. Action: The path specified in the *archive_file* column must be created prior to running the Report Generator task. Ensure a valid path has been resolved if a format specifier is used in the *archive_file* entry. Ensure the Shared domain user has permission for the specified file and directory.
Generator report *report_name*	Cause: A status message reporting that *report_name* is in the process of being generated. The Temporary file associated with *report_name* is present in the specified location. Action: None required.
Gen Info panel exceeds 2048 records per report	Cause: More than 2048 entries exist in the Information table for a given report. Action: Reduce the number of objects in the identified report file.
Invalid file size for CT file *report_name*	Cause: The CT entry associated with *report_name* is corrupt. Action: Run **ctgen -v2 rpt**. Note any errors and make corrections.
No Report Generator Information panel data	Cause: No information is in the Report Generator Information table. Action: Verify information is displayed in the Information table. Even if an object is not defined within the *format_file*, one entry should be entered.

Error Message	Cause and Action
No reports configured	Cause: No valid reports can be resolved. Action: Ensure the report generator tables are configured in the Shared domain. Force the regeneration of the CT file by running **ctgen -v2 rpt** noting any errors.
No trigger exists for a report	Cause: The table associated with *report_name* does not have an end trigger defined. Action: Declare at least an end trigger for *report_name*.
No triggers in CT file *report_name*	Cause: The table entries associated with *report_name* do not have the minimum requirements of an end trigger defined. Action: Declare at least an end trigger for *report_name*.
Out of RAM	Cause: The report generator can not obtain enough RAM to load the CT file and the associate format files. Action: Free system resources to ensure enough memory exists.
Printing report *report_name*	Cause: A status message reporting that *report_name* is in the process of being spooled to the Spooler task. The printed report should be generated if the Spool task is activated and all associated hardware is in working condition. Action: None required.
Reading CT file *report_name*	Cause: A status message reporting that the CT file for *report_name* is being loaded into memory. Action: None required.
Task initialization failed	Cause: Report Generator is unable to register with the kernel. Action: Start the Shared domain with a **-d** argument and note any errors as the Report Generator is starting.
Trigger table error	Cause: The watch list for trigger tags generated an internal error. Action: Regenerate the CT file using **ctgen -v2 rpt** and note any errors.

Chapter 21. . . .

Run-Time Manager

The Run-Time Manager (also known as Run Manager) allows you to start, monitor, and stop individual FactoryLink server tasks. This chapter describes how to configure and use the Run-Time Manager.

OPERATING PRINCIPLES

The Run-Time Manager task starts, stops, and monitors all other FactoryLink tasks according to the settings configured in the System Configuration table.

At system startup, Run-Time Manager reads the System Configuration table to determine which tasks to start, their start order, priority, debug status, and program arguments. There are several different ways to invoke Run-Time Manager, which will be discussed later in this chapter.

During run time, Run-Time Manager monitors the status of the other tasks in the system and updates system tags in the real-time database with that status information. A set of standard Run-Time Manager mimics is included in the FactoryLink Examples Application and Starter Application templates. These mimics may be edited, replaced, or used as-is. They are intended for administrators and maintenance people. The developer should consider limiting access to these screens for security purposes, since the screens can be used to shut down individual tasks or the entire application.

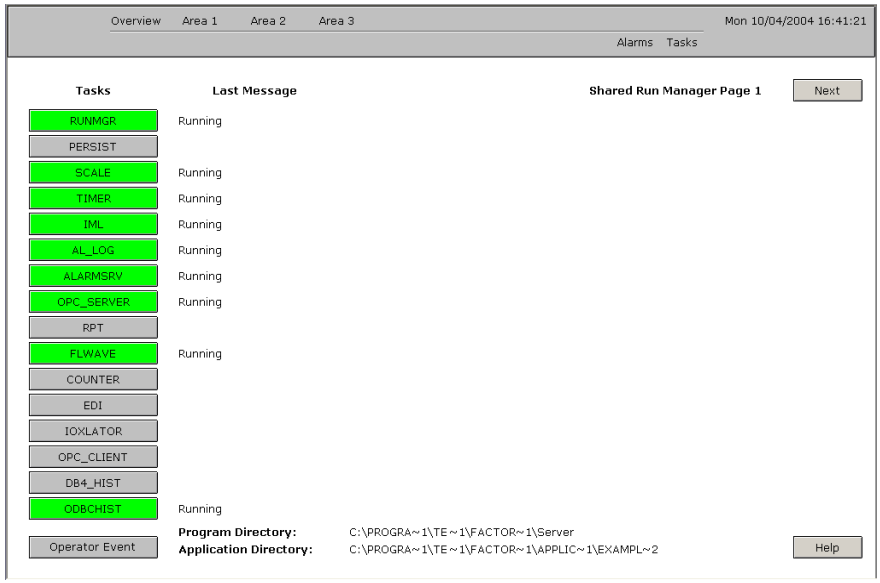
At system shutdown, Run-Time Manager reads the System Configuration table to determine the order in which to stop tasks and performs an orderly shutdown. It is important that you perform an orderly shutdown rather than just turning off the computer where FactoryLink is running.

FactoryLink comes with a pre-configured Run-Time Manager mimic as shown in Figure 21-1. Using Client Builder, you can customize or replace the Run-Time Manager mimic according to your needs.

ACCESSING THE RUN-TIME MANAGER MIMIC

1. Run the Examples Application or an application made using one of the FLNEW templates.
2. Click the **Tasks** button to display the Run-Time Manager mimic.

Figure 21-1 Run-Time Manager Mimic



There are Run-Time Manager mimics for both Shared and User Domains. The mimics have the following major components, not all of which appear on the first mimic:

- The task buttons on the left start and stop the tasks as well as indicate one of the following states:

Color	State	Color	State
Gray	Inactive	Blue	Starting
Green	Running	Yellow	Stopping
Red	Error has occurred		

- The Last Message displays the most recent error or system message.
- Program Directory is the Server's FLINK environment variable.
- Application Directory is the Server's FLAPP environment variable.
- Application Name is the Shared or User FLNAME environment variable.
- Application User is the Shared or User FLUSER environment variable.
- The Application Name Button on the second Shared Run-Time Manager mimic, controls the Application Start/Stop.

SYSTEM CONFIGURATION TABLE

Use the System Configuration Information table to define how Run-Time Manager controls tasks at system startup. FactoryLink sets default run-time parameters in the System Configuration Information table whenever you create a new application using FLNEW, one of the FLNEW Starter templates, or the Examples Application.

The default values establish the following parameters for the run-time FactoryLink system:

- Tasks that start up when the application is running
- Tasks allowed to run as foreground tasks
- Order in which tasks start up and shut down
- Priority of each task
- Domain associated with each task

Use this table if you want to make the following modifications:

- Change task settings, such as flags and program arguments
- Re-sequence the order in which tasks start up
- Change foreground and background task identification
- Add a new task to an existing application

Accessing

In your server application, open **System > System Configuration > System Configuration Information** in form view.

> **Note:** Even though you can open the System Configuration Information table in the Grid view, it is recommended that you open this table in the Form view.

Field Descriptions

Task Name Name of the process (task), such as RECIPE, ALOG, or TIMER. Do not modify these default names. For an external program, define a program name.

Valid Entry: alphanumeric string of 1 to 32 characters
Default: default settings vary for the particular task

Task Description Description of the task listed in the Task Name field. For example, the description of the SPOOL task is “Print Spooler.”

Valid Entry: alphanumeric string of 1 to 80characters
Default: default settings vary for the particular task

Task Flags Indicators to Run-Time Manager of how the task behaves at startup. A task can have multiple flags with flag values entered in any order. The entries and descriptions are as follows:

These flags can be edited directly if desired.

Valid Entry: Run at Startup, Create Session Window, Suppress Online Configuration, Suppress Task Hibernation

- S Create Session Window—Provides the task or process with its own window. Any output to that task is directed to this window rather than to the Run-Time Manager window. RUNMGR, RTMON, and ECS GRAPHICS require their own windows.
- R Run at Startup—Activates this task at FactoryLink startup. To allow a task to be started manually by an operator, do not enter the R flag.
- O Suppress Online Configuration—Suppresses Online Configuration for this task.
- H Suppress Task Hibernation—Suppresses hibernation for this task.
- F Foreground flag—Puts this task in the foreground at startup. Use the F flag if the task has neither the S nor the R flags.

Default: default settings vary for the particular task

Start Order Indicates the order in which tasks should start at run time. The task defined with Start Order 0 starts first, Start Order 1 starts next, and so on. The number 0 is reserved for RUNMGR. Tasks with the same start order number will start consecutively.

Use the following guidelines to determine the Start Order for certain tasks:

Set the Start Order for...	To start...
Historian	before Logger
Logger	before Trending
Math & Logic	coordinated with the procedures. For example, if a procedure is dependent on data from an external device, start the driver before Math & Logic.

Valid Entry: numeric value from 0 to 31

Default: default settings for the particular task

Start Priority

Processing priority for the task. Unless you are knowledgeable and experienced in setting priorities in the operating system, you should leave the priority at the default value.

Do not set any FactoryLink task to priority class 0 or 1. Use caution in assigning a task with priority class 3 because a time-critical task takes priority over a foreground task. A foreground task takes priority over regular tasks at run time.

The priority is a three-digit hexadecimal value which is divided into two parts:

First part (the 2 in 201): Hex value that specifies the operating system class of priority as listed below. This part is inactive in Windows.

Valid Entry: one of the following hex values:

- 0 Current class unchanged
- 1 Idle
- 2 Regular (default)
- 3 Time-critical

Second part (the 01 in 201): Two-digit hexadecimal value (00 to 1F [0 - 31 decimal]) that specifies the priority within the priority class listed above. The higher the number, the higher the priority within the class.

Valid Entry: Hexadecimal value 00 to 1F
Default: 01

Executable File

Indicates the name and location of the executable file for the task. It is recommended that you not change the default. If this is a relative path name to FLINK, do not use leading spaces.

Valid Entry: Any path name which has the following format:
`\DIRECTORY\SUBDIRECTORY\FILENAME`
Default: default settings for the particular task

Program Arguments

These arguments tell the task to run in a special way. For more information about program arguments, see each task in this guide.

Valid Entry: valid program argument
Default: default settings for the particular task

If you view the System Configuration Table in Grid format, you will see additional fields that contain information about the task at run time. This information can be displayed on the Run-Time Manager screen or another screen if desired.

Start Trigger Tag that triggers the task. If the Flags field contains an R for this task, at run time a 1 (ON) is written to this tag. This causes FactoryLink to start the task automatically. If the Flags field for this task does not contain an R, the task does not start until the operator writes a 1 (ON) to this tag by clicking on the task name on the Run-Time Manager screen.

Valid Entry: tag name
Default: default name for the particular task
Valid Data Type: digital or analog

Task Status Tag that contains the status of the task. This tag is updated by the Run-Time Manager task and can have the following values:

0 Inactive
1 Active
2 Error
3 Starting
4 Stopping

Valid Entry: tag name
Valid Data Type: analog
Default: default name for the particular task

Task Message Tag to which the task listed in the Task Name field writes any run-time messages. These messages appear in the Message column of the Run-Time Manager screen.

This tag can have the following message values:

Inactive	Active	Error
Starting	Stopping	

Valid Entry: tag name
Valid Data Type: message
Default: default name for the particular task

Display Status Tag that contains a text version of the status of the process. The task status is displayed in the Status column on the Run-Time Manager screen.

Valid Entry: tag name
Valid Data Type: message
Default: default name for the particular task

Display Name	Tag that contains the string entered in the Task Name field. This task name is displayed in the Task field on the Run-Time Manager screen. Valid Entry: tag name Valid Data Type: message Default: default name for the particular task
Display Description	Tag that contains the string entered in the Description field of this table. Valid Entry: tag name Valid Data Type: message Default: default name for the particular task
Application Directory	This field is reserved for future use.
Program Directory	This field is reserved for future use.
Program Arguments	Values used as the arguments to the process. If this field is blank, no arguments are passed to the task. Program arguments are not case-sensitive.

EDITING TASK INFORMATION

The application utility, FLNEW, establishes default values for tasks in the System Configuration Information table and associates each task with a specific domain. These defaults and associations are based on evaluation of task performance and are recommended for most applications.

You may edit this table to identify an external program to the system. Although you can make changes in some fields, it is better not to change any fields except Flags and Display Status.

ADDING NEW TASKS

The Run-Time Manager uses pre-defined tags and array dimensions to automatically display items on screen. These tag names and array dimensions appear in the System Configuration Information table for each task displayed on the Run-Time Manager screen. If you add another task to the Run-Time Manager screen, use the next available array dimension.

The next available array dimension is determined by a task's position in the display sequence. If you view the System Configuration table associated with the Shared domain, you will find all tag names associated with Run-Time Manager end with a dimension of [0], Persistence tag names end with a dimension of [1], Scale tags end in [2], and so on. For example, the complete entry in the Task Status field for the Run-Time Manager task is TASKSTATUS_S[0].

If the information in a field is longer than the number of characters that fit in the allotted space on the screen, part of the entry will scroll out of sight.

To see characters that have scrolled out of sight, press the → and ← keys. The field scrolls to display the text. The bracketed number represents an array dimension.

Complete the following steps to add a task to the Run-Time Manager screen:

1. Choose the appropriate domain for the task.
2. Open the System Configuration Information table.
3. Starting under the last row of information in the System Configuration Information table, add the required information about the new task to each field. Use the Copy and Paste functions to copy duplicate information, such as tag names, from the previous row.
4. Review the previous task in the task list to determine its dimension. Assign the new task's tag names the next available array dimension. If you used the Copy and Paste functions to copy information from an existing row, modify each array dimension to be the correct value.
5. Save the information when the table is complete.

The next time you run the application, the new task and its related information is displayed at the bottom of the specified domain's Run-Time Manager screen.

ARCHIVING ERROR MESSAGES

This section contains information about setting up FactoryLink to create error message log files. Whenever an error occurs in a FactoryLink task at run time, FactoryLink sends a message for display to the Run-Time Manager screen. FactoryLink also sends a longer, more descriptive message to the log file (if the file is set up). The following FactoryLink tasks can create an error message log file:

- Batch Recipe
- Database Logger
- Database Browser
- FactoryLink Local Area Networking (FLLAN)
- File Manager
- Any Historian
- Math & Logic (Interpreted mode only)
- Statistical Process Control (SPC Log, SPC View, and SPR)
- Trending

You can configure FactoryLink to automatically create an error message .LOG file at startup.

In Windows XP and 2000, .LOG files are stored in the FLAPP/FLNAME/FLDOMAIN/FLUSER/LOG directory.

where

FLAPP	is the environment variable for the application directory.
FLNAME	is the environment variable for the application name.
FLDOMAIN	is the environment variable for the domain.
FLUSER	is the environment variable for the user name.

FactoryLink creates the log file name using the following format:

```
XXMMDDYY.LOG
```

where

XX	indicates the FactoryLink task.
MM	is the month of the year (1-12).
DD	is the day of the month (1-31)
YY	is the year since 1900 (00-99).

If you specified during installation you wanted to install the Old version of FLLAN, FLLAN's .LOG files will have the following path and file names: FLAPP\NET\FLLANSND.LOG and FLAPP\NET\FLLANRCV.LOG

If you configure FactoryLink to create a log file for a task, FactoryLink logs a message in its log file whenever that task generates an error. The messages in the log file are more descriptive than those that appear on the Run-Time Manager screen.

For debugging purposes, configure FactoryLink to create log files automatically at startup. Complete the following steps to configure FactoryLink to do this:

1. Open the System Configuration Information table.
2. Ensure the current domain selected. Locate the corresponding entry for the task in the Task Name field.
3. Place the cursor on the corresponding entry.
4. Tab to the **Program Arguments** field.
5. Enter **-L**, **-V#** (not case-sensitive) where # is 2, 3, or 4 in the Program Arguments field. The greater the number, the more information you receive. (Enter -L, -D# where # is any number from 2 to 22 for the File Manager and FLLAN tasks.)
6. Choose **Enter** to save the information.
7. Repeat steps 2 through 6 for each task that needs a log file.

The log files continue to grow at run time as messages are logged to them until the operator shuts down and restarts each task. Then, FactoryLink creates new log files. However, FactoryLink creates only one log file per task per day no matter how many times each task is shut down and re-started in one day.

Delete old log files periodically to prevent log files from using too much disk space. You can configure the File Manager task to delete files for you. For example, File Manager can delete them each day at midnight or when the files specified reach a specified size.

Caution: Do not delete the current log file if the task is still running. This causes errors.

When you are finished debugging your application, you can remove the Program Arguments from the System Configuration Information table to eliminate the creation of extra files.

STARTING SERVER APPLICATIONS

How you start the Run-Time Manager depends on your preferences and desired use of the system.

Using the FactoryLink Application Manager	On the server, open **Start > Program Files > FactoryLink > FactoryLink Application Manager.** This method would typically be used by developers or administrators.
Using the FactoryLink Configuration Explorer	Right-click on any server application and click **Start/Stop > Start**. This method would typically be used by developers or administrators.
Using the Autostart feature	Right-click on any server application and click **Start/Stop > Autostart**. FactoryLink starts the selected application during the boot of a FactoryLink server machine before the user logs in. The application is started as an NT Service. This method would typically be used by operators on run-time only systems.

TUNING KERNEL MEMORY

FactoryLink mailbox messages enable tasks to communicate between themselves via a queue as opposed to the standard FactoryLink single-value/change notification system. The queue removes the chance of the receiving task missing data should changes occur rapidly. Since all writes to a mailbox are stored in the kernel until read, kernel memory resources can be exhausted when the messages written into the kernel are not read quickly enough by their target consumer task. Once these resources are exhausted, operations requiring mailbox communication, such as screen changes or database logging, no longer function.

Once a mailbox has been stuffed with orphaned messages, no other mailbox writes can be performed, even if these writes are to a different mailbox.

The system must be able to be tuned to handle large quantities of mailbox messages as well as not allow any mismatched mailbox producer/consumer task combinations exhaust the kernel of all resources.

To make this configuration tunable, a switch to the Shared domain instance of the Run-Time Manager is used. This switch is configured in the System Configuration Information table. The syntax of this switch is

-m<max_seg>[:<max_mbxsegs>[:<max_onembx>]]

where

<max_seg>	Maximum number of kernel segments. (default = 1000 of 64K bytes each)
<max_mbxsegs>	Maximum segments used for mailbox messages. (default = 250)
<max_onembx>	Maximum number of K bytes of message space held by one mailbox. The default sets no per-mailbox ceiling.

Without the -m switch, the system uses the default settings.

In addition to system-wide limits, a memory usage ceiling can be set per mailbox tag. The message length field of the Object table, currently supported for message tags, can be set with a maximum memory usage for its associated mailbox tag. This is specified in K bytes. The per-mailbox limit supersedes the system-wide mailbox limit.

Once a mailbox tag ceiling is reached, all subsequent writes to that tag are dropped. A new error code, FLE_MBXLIM_EXCEED, is returned for this case.

The per-mailbox K byte limit can also be set or obtained through the Programmer's Access Kit (PAK).

PROGRAM ARGUMENTS

The options specified in the Run-Time Manager apply to all tasks started for the application. You can set options for individual tasks that override these defaults using the System Configuration Table.

Where you define the default run-time options depends on whether you are starting the Run-Time Manager from a FactoryLink icon or from an operating system command line.

Argument	Description
`-d`	Turns on debug mode. Any errors encountered are logged to the log file. If you specify this option, you can use Ctrl+C to shutdown Run-Time Manager.
`-a<flapp_dir>`	Defines the full path of the directory containing the application files. This path overrides any path set by the FLAPP environment variable.
`-p<flink_dir>`	Defines the full path of the directory containing the FactoryLink programs. This path overrides any path set by the FLINK environment variable.
`-fl`	PID check. If experiencing kernel lock-up problems, the switch adds extra checking to prevent rogue tasks from corrupting the kernel; however, there is a performance penalty.
`-L`	Logs errors and other data to a log file
`-t<timeout>`	Defines the start/stop time-out, in seconds, for the Run-Time Manager error report process. The default time-out is 60 seconds.
`-s`	Starts only the shared domain on a PC platform. The user domain is not started.
`-n<fldomain>`	Defines the domain name, where *domain* can either be shared or user. If you specify shared, only the shared domain is started. This overrides the FLDOMAIN environment variable.
`-i<flname>`	Defines the name of the application to start. This overrides the FLNAME environment variable.
`-u<fluser>`	Defines the user name. This overrides the FLUSER environment variable.
`-w`	Turns on the warm start mode. If you specify this option, FactoryLink loads persistent tags with the last value saved for them.

ERROR MESSAGES

Error Message	Cause and Action
Bad command *number*	Cause: An invalid command was written to the global tag COMMAND. Action: Verify the external process compatibility with FactoryLink.
Bad file size for *filename*	Cause: The FLAPP/CT/TYPE.CT file is damaged. Action: Delete the file and restart the application to rebuild the file.
Bad index in file *filename*	Cause: Unable to read the index of the FLAPP/CT/TYPE.CT file. Action: Delete the file and restart the application to rebuild the file.
Can't convert subgroup table name	Cause: Tried to convert an older SPC application. Action: See the conversions instructions in the *Conversion Guide*.
Can't create appl. *filename* error *error number*	Cause: Tried to start an application already started. Action: See “Error Numbers and Keywords” on page 498.
Can't open file *filename*	Cause: The disk may be full, or the log file may be opened by another process. Action: Delete unnecessary files. If this error occurs often, additional disk space may be required.
Can't read options file *FL.OPT*	Cause: FL.OPT is damaged. Action: Delete the fl.opt file and rerun the License Wizard.
Can't start process *process name*	Cause: The binary file may not exist, may not be executable, or the filename in the EXECUTABLE FILE field in the System Configuration Table may be incorrect. Under some system platforms this may indicate an insufficient number of processes. Action: Verify the correct file name in the System Configuration Table.

Error Message	Cause and Action
Can't stop process *process name*	Cause: The process may already have stopped. Action: Attempt to stop the process manually if it has not stopped.
Client *processes* failed to start	Cause: One or more processes could not be started. Action: See the error messages for the particular process.
Client *processes* failed to stop	Cause: The process may already have stopped. Action: Attempt to stop the process manually if it has not stopped.
Directory *directory name* does not exist	Cause: Cannot find the FLAPP directory specified by the environment variable. Action: Set FLAPP to a valid application directory.
Directory *directory name* is not a valid FactoryLink directory	Cause: FLAPP does not have a valid subdirectory structure. Action: (1) Check the FLAPP directory. Add the missing subdirectories. (2) Set FLAPP to a valid application directory.
Domain *domain name* can only have one instance	Cause: A “parent” domain is configured as having more than one instance. Action: A “parent” domain may have only one instance. Open the Domain tag List and change the entry in the #INST field to 1.
Domain *domain name* isn't in the domain CT	Cause: The specified domain does not exists in the domain.CT. Action: Verify the domain name exists in the Domain Tag List and is entered correctly.
Environment Tag *tag name* has an invalid type	Cause: The wrong data type was specified for the field. Action: Enter the correct data type for this field.
FactoryLink initialization failed	Cause: FactoryLink system failed to initialize. Action: Verify the license is enabled. Use the License Wizard to see the purchased options.
FactoryLink system monitoring failed	Cause: The system was unable to start the monitor task. The system may not contain sufficient memory. Action: Stop unnecessary processes.

Error Message	Cause and Action
Grace period has expired. Software must be registered	Cause: FactoryLink was installed without registering it within the 10 day grace period. Action: You need to register your FactoryLink software.
Kernel initialization failed	Cause: FactoryLink was already initialized. FactoryLink was already running when startup was attempted. Action: No action required
Monitor tag array definition failed	Cause: A System Configuration table monitor tag (Start Trigger, Task Status, or Task Message) may be undefined or defined incorrectly for one of the tasks being started. Action: Open the System Configuration table and define the tags correctly.
Number of defined proc *process number* more than max *maximum number*	Cause: More than 31 processes were started. Action: Start fewer processes.
Out of RAM	Cause: No more memory is available. Action: Close any unnecessary windows or programs. Add more memory to the system if this error occurs often.
Output tag array definition failed	Cause: A System Configuration table output tag (Display Status, Display Name, or Display Description) may be undefined or defined incorrectly for one of the tasks being started. Action: Open the System Configuration table and define the tags correctly.
Process *process name* may not have started	Cause: The process failed to become active. It may not have been able to register with FactoryLink. Action: Verify the filename in the EXECUTABLE FILE field in the System Configuration table. Verify the process compatibility with FactoryLink.
Process *process name* may not have stopped	Cause: The process may already have stopped. Action: Attempt to stop the process manually if it has not stopped.

Error Message	Cause and Action
Read failed on file *filename*	Cause: A read operation on the named file failed. Action: Verify configuration table entries, communication parameters, hardware identification information, and electrical connections.
Read header failed on file *filename*	Cause: The .CT file may be damaged. Action: Delete the .CT file in the FLINK/ct directory and restart the application to rebuild the file.
Real-time database isn't initialized	Cause: FactoryLink is starting or shutting down. Action: Start FactoryLink again.
Real-time database for application doesn't exist	Cause: An attempt was made to start a User domain when the Shared domain did not exist. Action: Start the Shared domain.
Run-Time Manager: errno = *error number*	Cause: The Run-Time Manager was awakened when there was nothing for it to do. (Internal error.) Action: "Error Numbers and Keywords" on page 498
Run-Time Manager CT processing failed	Cause: One or more of the tags ARGUMENT, COMMAND, PASSWORD, SHUTDOWN, or STARTUP is not defined in the GLOBAL.CDB file. The file is damaged or installed incorrectly. Action: Contact your technical support representative.
Run-Time Manager failed to start	Cause: The Run-Time Manager must be the first task started. The Start Order number for RUNMGR must be 0; no other task may have a zero for a Start Order number. Action: Check the System Configuration table to verify the Start Order.
Run-Time Manager failed to stop	Cause: The process may already have stopped. Action: Attempt to stop the process manually if it has not stopped.
Run-Time Manager is already running	Cause: A copy of the Run-Time Manager is already running for the Domain and User name specified. Action: Change the Domain or User name.

Error Message	Cause and Action
Software has not been enabled	Cause: An error has occurred with the f1.key file in the opt directory. Action: Verify the option key is installed and the license is enabled. Use the License Wizard to see the purchased options.
Write error on log file *filename*	Cause: The disk may be full. Action: Delete unnecessary files.

Error Numbers and Keywords

Each error number has a corresponding error keyword, which does not display in the error message. Knowing the error keyword helps Customer Support engineers identify the cause of the problem.

Keyword	Cause and Action
1 FLE_INTERNAL	Cause: Internal Error Action: See “Correcting Internal Errors” on page 502.
2 FLE_OUT_OF_MEMORY	Cause: Not enough RAM is available. Action: Add RAM to the system.
4 FLE_NO_FLINK_INIT	Cause: The real-time database has not been created or cannot finish initializing. Action: Shut down FactoryLink and restart it.
5 FLE_NO_PROC_INIT	Cause: The task name of the specified task is not entered in the System Configuration Table. Action: Enter the task name in the Task Name field of the System Configuration Table. Cause: The task is already running. Action: Nothing—the task will continue to run. Cause: The indicated task is not enabled in the FactoryLink key. Action: Contact your FactoryLink sales representative to obtain the proper key and/or option.

Keyword	Cause and Action
7 FLE_BAD_ARGUMENT	Cause: Internal error—a task tried to pass an invalid argument to the real-time database. Action: See “Correcting Internal Errors” on page 502.
9 FLE_BAD_TAG	Cause: The .CTs need to be rebuilt. Action: Rebuild the .CTs by running CTGEN.
10 FLE_NULL_POINTER	Cause: Internal error—a task gave out a null pointer. Action: See “Correcting Internal Errors” on page 502.
12 FLE_PROC_TABLE_FULL	Cause: 31 tasks are running in the chosen domain. Only 31 tasks can run in each domain at once. Action: No action required—do not try to run more than 31 tasks per domain at a time.
13 FLE_BAD_PROC_NAME	Cause: The .CTs need to be rebuilt. Action: Run CTGEN to rebuild the .CTs. Cause: A task name used in the application is not entered in the Task Name field of the System Configuration Table. Action: Open the System Configuration Table and ensure the names of all tasks in the application are present in the Task Name field. Cause: The requested task is not running. Action: Ensure you started all tasks needed to run the application.
14 FLE_BAD_USER_NAME	Cause: The FLUSER environment variable is not set. Action: Set the environment variables; restart FactoryLink.
22 FLE_ALREADY_ACTIVE	Cause: The task is already running. Action: None-do not start a task already running.
23 FLE_NOT_LOCKED	Cause: Internal error—a task tried to unlock the real-time database without having locked it first. Action: See “Correcting Internal Errors” on page 502.

Keyword	Cause and Action
24 FLE_LOCK_FAILED	Cause: Internal error—a task used an invalid Task ID to lock the real-time database. Therefore, the task did not successfully lock the real-time database or perform its function. Action: See “Correcting Internal Errors” on page 502.
25 FLE_LOCK_EXPIRED	Cause: A task has kept the real-time database locked longer than the kernel allows. Action: None—currently, the kernel allows tasks to lock the real-time database for as long as necessary. Therefore, the lock time will not expire.
26 FLE_WAIT_FAILED	Cause: Internal error—while trying to do a change-wait on a tag, a task sent an invalid Task ID to the real-time database. Action: See “Correcting Internal Errors” on page 502.
28 FLE_QSIZE_TOOBIG	Cause: Internal error—a task attempted to attach a queue to a tag, but there was not enough memory. Action: See “Correcting Internal Errors” on page 502.
29 FLE_QSIZE_CHANGED	Cause: Internal error—a task attempted to attach a queue to a tag, but a queue of a different size was already attached. Action: See “Correcting Internal Errors” on page 502.
30 FLE_NO_TAG_LIST	Cause: This error only occurs on custom tasks written using the FactoryLink PAK. If the custom task was set up to access tags by name instead of by ID number, this error can occur if the tag list has not been defined. Action: Use the API function FL_SET_TAG_LIST to define the tag list. Then, restart FactoryLink. See the *Programmer’s Access Kit (PAK)* guide for information about L_SET_TAG_LIST.
31 FLE_TAG_LIST_CHANGED	Cause: This message can be displayed only if an application has been set up so one task monitors another task’s tag list. If one task modifies the other task’s tag list, the task that modified the list will return this message, thus informing the task that its list has been modified. Action: No action required

Keyword	Cause and Action
32 FLE_WAKEUP_FAILED	Cause: Internal error—The real-time database tried unsuccessfully to wake a task waiting for the value of a particular tag to change. Action: See “Correcting Internal Errors” on page 502.
33 FLE_NO_SIGNALS	Cause: If a task calls the FL_RECV_SIG function to find out if any signals have been sent to it, the FL_RECV_SIG function returns a message. If no signals have been sent to the task, FL_RECV_SIG returns this message. Action: No action required
34 FLE_SIGNALLED	Cause: The real-time database sends this message to inform a sleeping task that the task has received a signal, not a change in value of the tag for which it is waiting. Action: No action required
35 FLE_NOT_MAILBOX	Cause: The .CTs need to be rebuilt. Action: Run CTGEN to rebuild the .CTs. Cause: While configuring a task, you entered a tag of a data type other than MAILBOX in a field that requires a MAILBOX tag. Action: Open the configuration tables of the task containing the error. Ensure the data types of the tags in all fields requiring MAILBOX tags are defined as MAILBOX.
36 FLE_NO_MESSAGES	Cause: If through the PAK QUERY_MBX function, a task requests to view a message whose queue number falls outside the range of available messages, the QUERY_MBX function returns this message. For example, if a task requests to view the third message waiting in the queue, and only two messages are in the queue, then the task is requesting to view a message outside the range of available messages. Action: No action required

Keyword	Cause and Action
37 FLE_ACCESS_DENIED	Cause: If a task tries to read a mailbox tag that it is not allowed to read, the real-time database returns this error. If the task is a custom task developed using PAK, the programmer may not have set up ownership of the mailbox tag by the task requesting to read from it. Action: Use the PAK function FL_SET_OWNER_MBX to establish the task as owner of the mailbox tag. Cause: Internal error Action: See “Correcting Internal Errors” on page 502.
41 FLE_APP_EXISTS	Cause: A real-time database with the same FLNAME but a different number of users or tags already exists. Action: Either shut down the running real-time database or set FLNAME with a different name.
42 FLE_NO_FLINK_RTDB	Cause: A user domain was started for a real-time database for which the shared domain has not been started. Action: Start the shared domain first.

Correcting Internal Errors

Internal errors are generally caused by one of the tasks in the system in use and not by your application. You can use the following guidelines to correct an internal error:

1. Try to determine which task is sending the error by shutting down FactoryLink, restarting it, and starting each task, one at a time.
2. Write down any error messages displayed on the Run-Time Manager screen and their corresponding tasks. (The task having the problem may generate a seemingly unrelated error message.)
3. Contact the supplier of the task if the task in error is an external task.
4. Contact your technical support representative if the task in error is a FactoryLink task.

Chapter 22. . . .

Scaling and Deadbanding

The Scaling and Deadbanding task converts or scales incoming raw data to a value in a more useful format using a linear relationship. Scaling is often referred to as engineering units conversion. The task can also indicate a deadband or area around a scaled value that is small enough to be considered insignificant and is ignored.

Many values read from various types of control equipment are in units other than those the user wishes to display, manipulate and/or archive. The Scaling and Deadbanding task eliminates the need to process data through an intermediate routing mechanism and the need to write code to perform the scaling function when the scaling is linear. If given ranges for the incoming and desired data values, it can derive the necessary conversion factor and/or offset and perform the linear scaling calculations automatically using the formula:

y = mx + b

where **x** is the raw value, **m** is the multiplier, **b** is a constant, and **y** is the result.

If you indicate a deadband around a value, the new value is stored and a new deadband recalculated, but the new value is not written to the real-time database. Since FactoryLink tasks process values upon every change, deadbanding provides a means of saving processing time and improving system efficiency.

> **Note:** The deadbanding portion of the function cannot be implemented without configuring the scaling portion of the function.

OPERATING PRINCIPLES

The scaling function only applies for tags with an analog, longana, or float data type.

Scaling is configured using a pair of ranges for raw values and a pair for scaled values. These ranges can be specified as constants or tags. The scaling formula is adjusted accordingly if one or more of the range tags changes.

When a value is written to a raw value tag, its related scaled value tag is updated accordingly. This is a *raw-to-scaled* conversion.

When a value is written to a scaled value tag, its raw value tag is updated accordingly. This is a *scaled-to-raw* conversion.

Prior to changing a range tag, raw value tag, or scaled value tag, the function should be disabled using the Scaling Lock Tag. When the Scaling Lock Tag has a nonzero value, changes

made to the tag are not propagated to their related members. After the changes to that function are made and the function is re-enabled, the current raw value is scaled and written to the scaled value tag. Any changes to the ranges are applied to the scaled value as well.

Deadbanding applies to raw-to-scaled conversion but not to scaled-to-raw conversion, and may be specified in one of two ways:

- As an absolute (ABS) number of Engineering Units (EUs)
- As a percentage (PCT) of the scaled range

During raw-to-scaled conversion, a newly calculated scaled value that does not exceed the deadband is not written to the database. If deadbanding is being applied to a tag associated with scaling rather than a specific alpha-numeric range, deadbanding is specified by a percentage of a range rather than as an absolute value. If the deadband variance for a scaled tag is specified as an absolute value, then no deadbanding is applied to the associated raw tag.

SCALING AND DEADBANDING INFORMATION TABLE

The Scaling and Deadbanding Information table sets up the value ranges required to scale a value coming into the real-time database. These ranges can be either constants or tags.

Accessing

In your server application, open **Scaling and Deadbanding > Scaling and Deadbanding > Scaling and Deadbanding Information.**

Field Descriptions

Scaled Tag	Tag to which scaled values will be written.
Raw Tag	Tag from which the field raw values are read.
Minimum Raw Value	The lowest value for raw data. Either a constant value or a tag can be specified in this field.
Maximum Raw Value	The highest value for raw data. Either a constant value or a tag can be specified in this field.
Minimum Eng. Unit	The lowest value for scaled data. Either a constant value or a tag can be specified in this field.
Maximum Eng. Unit	The highest value for scaled data. Either a constant value or a tag can be specified in this field.

Deadband Value	The amount, in either absolute value or percent of total value that, when applied, creates a range on either side of the value the recalculated value does not have to be written to the database in. If a tag name is specified for the deadband value rather than an integer, no deadbanding occurs until a value is written to the specified tag from the database.
Deadband Abs./Pct.	Enter Absolute or Percentage to indicate if the deadband value specified is an absolute number of engineering units (EUs) or a percentage of the scaled value range.
Scaling Lock Tag	Use this field to temporarily disable the scaling feature for the scaled tag. When the Scaling Lock Tag has a nonzero value, changes made to the tag are not propagated to their related members. After the changes to that function have been made and the function is re-enabled, the current raw value is scaled and written to the scaled value tag.

CONFIGURATION TECHNIQUES AND CONSIDERATIONS

Shortcut Configuration Example

When you enter scaling information using the Tag Editor instead of the Scaling and Deadbanding Information table, additional tags are automatically created for raw and scaled values for the tag. The new tag names all have the “root” of the tag you created. The task appends the suffixes .raw, .rawmin, .rawmax, .eumin, .eumax, .dead, and .lock to create seven additional unique tags for each value with the appended suffixes.

For example, try the following steps:

1. Create a tag in the Math and Logic Variables table named **scale_test,** then **Save**.
2. With scale_test still selected, open the Tag Editor and select the Scaling/Deadbanding tab.
3. Enter High and Low values for the Raw and Engineering Units fields. This will establish the necessary information to calculate the linear conversion. For example, to convert Fahrenheit to Celsius, enter the values in the following table:

Linear Scaling	Raw	Engineering Units
High	212	100
Low	32	0

4 Enter a Disable Tag and Deadbanding Value if desired, then press **OK**.

5 Open the Scaling and Deadbanding Information table in grid view. You will see that new tags have been added. The tag names all have the "root" of **scale_test**. The scaling task appends the suffixes **.raw**, **.rawmin**, **.rawmax**, **.eumin**, **.eumax**, **.dead**, and **.lock** to create seven unique tag names for each value.

6 Examine these tags in the Tag Editor and you will see that the default values are the values you entered in the Scaling/Deadbanding tab.

Scaling of Persistent Tags

If the Scaled Tag has persistence configured when you're using the Tag Editor, you automatically create the .raw tag and all other "." scaling tags with the same persistence type as the Scaled Tag.

If you enter the scaling data manually into the table, you need to manually add persistence to the .raw tag.

Scaling and Drivers

Driver Type	Configuration Scenario
EDI	Use the Scaled Tag in the EDI driver information tables. FactoryLink automatically substitutes it with the .raw tag.
RAPD using IOXLATOR	Use the Scaled Tag in the IOXLATOR driver information tables. FactoryLink automatically substitutes it with the .raw tag.
OPC_CLIENT	Use the .raw in the OPC_Client tag definition table as the Data Tag

ERROR MESSAGES

Error Message	Cause and Action
Error reading CT record	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
Error%d: Unknown signal%d received	Cause: This is a FactoryLink internal error. Action: Operation is unaffected by this error.
Insert tag into DTP tag list failed	Cause: This is a FactoryLink internal error. Action: Operation is unaffected by this error.
Invalid deadband tag	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
Invalid deadbanding lock tag	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
Invalid maximum raw tag	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
Invalid maximum scaled tax	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
Invalid minimum raw tag	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
Invalid minimum scaled tag	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
Invalid raw tag	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
Invalid scaled tag	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
Invalid scaling lock tag	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No CT	Cause: The file SCALE.CT is missing Action: Rebuild SCALE.CT with CTGEN.

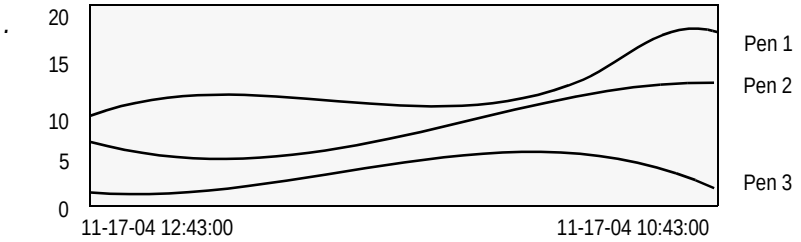
Error Message	Cause and Action
No CT table #0	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No deadbanding lock tag or value	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No deadband tag or value	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No maximum raw tag or value	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No maximum scaled tag or value	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No minimum raw tag or value	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No records in CT table #0	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No minimum scaled tag or value	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No raw tag or value	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No scaled tag or value	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No scaling lock tag or value	Cause: The file SCALE.CT is corrupt. Action: Rebuild SCALE.CT using CTGEN.
No tables in CT	Cause: The file **SCALE.CT** is corrupt. Action: Rebuild **SCALE.CT** using CTGEN.
Out of memory	Cause: The system does not have sufficient memory to run the process. Action: Free up memory by halting other running programs on the system and try running FactoryLink again.

Error Message	Cause and Action
Record%d: FPE 0x%04X occurred	Cause: A problem may exist with the hardware Floating Point Processor on the system. Action: Have the Floating Point Processor evaluated.
Record%d: Raw range is zero	Cause: The record cannot be processed because the minimum and maximum raw values are equal, resulting in a range of 0(zero). Action: Change one of the raw values to create a range greater than 0.
Record%d: Raw value is outside range	Cause: The raw data value received is less than the minimum or greater than the maximum specified. Scaling will continue, but the scaled value will be outside the range. Action: Change the raw value range to allow encompassing values outside the range.
Record%d: Scaled range is zero	Cause: The record cannot be processed because the minimum and maximum scaled values are equal, resulting in a range of 0 (zero). Action: Change one of the scaled values to create a range greater than 0.
Recorded: Scaled value is outside range	Cause: The scaled data value calculated is less than the minimum or greater than the maximum specified. Action: Change the scaled value range to allow encompassing values outside the range.

Chapter 23 . . . .
Trending

Using the Trend module, you can create animated graphs called trend charts that show numeric data graphically. Trend charts are capable of plotting a single value in a chart or multiple data points concurrently. The lines or bars on the chart are referred to as pens. A sample chart is shown in Figure 23-1.

Figure 23-1 Sample Trend Chart



OPERATING PRINCIPLES

The Trending module is composed of a Trend Server, a relational database, and two trend controls. The components work together with the logger and historian tasks to format real-time or historical information into a Trend chart that can be viewed at run time.

As data is collected or computed by FactoryLink, it is stored as a tag in the real-time database. Each time data is collected, or computed, the new data overwrites the value for the tag. To keep a history of the data, you must store it in a relational database.

The logger reads the value of specific tags in the real-time database and maps the tags to columns in a relational database table. The logger sends the data from the real-time database to the database via a historian mailbox. The historian inserts the data into the relational database. Once in this database, the data can be used by other applications.

The trend controls are used to view the data at run time. Depending upon your application needs, you should select one of the two controls:

- The Real-time Trend Control provides a quick and easy way to insert a real-time trend chart into a mimic. Use the Real-time Trend Control if you want to trend only real-time OPC data.
- The Historical and Real-time Trend Control lets you configure trend charts to display historical and real-time data or data from non-FactoryLink database tables.

Trending Functionality

The Trending module is very flexible and lets you design trend charts for various needs. A few of the major functions are described below.

For more information about using the trend controls, see the *Client Builder Help*.

Appearance

Most aspects of the appearance of a trend chart can be configured, such as size, background color, captions, text color, font, legends, line styles, and others.

Run-time Permissions

You can design your trend charts so that operators may have a large range of permissions for online changes at run time or none, depending upon your application. A few of the functions that can be permitted at run time are adding tags, viewing pen statistics, and changing pen appearance.

Multiple Pens

Using FactoryLink, you can create different types and numbers of pens. You can configure fixed pens at design-time to allow you to permanently assign a database table column to a particular pen. You can assign the column to a pen at run time, and you can assign multiple pens to a Trend chart at both design and run time.

> **Note:** We recommend you configure no more than eight pens to a chart for good readability.

Multiple Axes

You can configure multiple pens in a trend chart. FactoryLink creates an X and Y axis to correspond to each pen as each pen is created.

Panning & Zooming

Because all historical Trend data is written to a relational database, you can arrange it to show different data ranges. These ranges can be expanded or collapsed as needed.

Panning allows you to select the time span of data to display in a historical trend chart. Using this feature, you can move forward or backward through historical data and you can move to a specific time or sample.

Zooming is the ability to look at small or large chunks of data by changing the chart duration. Zooming either increases or decreases the amount of data displayed.

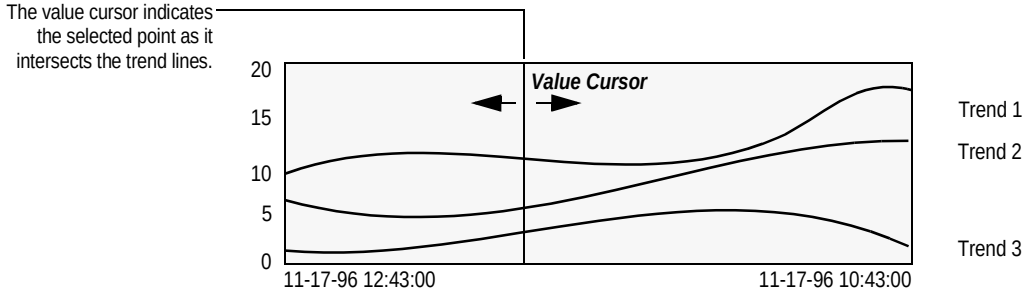
Tooltip Information

When you hold your cursor over a point in the Trend chart, information about that value appears in a text box over the point. Tooltip information includes the name of the pen, the value of the X-field, the value of the Y-field, and ID. The ID field shows information about the point in the ID/Key field of the database, if the database contains this field.

Value Cursor

A value cursor allows you to display the value associated with a point on a Trend chart. When you click anywhere in the chart at run time, the value cursor, which looks like a vertical bar, is displayed. You can write custom programs for pen cursor values. Figure 23-2 illustrates an example of a value cursor.

Figure 23-2 Value Cursor



Delta T

Delta T refers to an offset in time. In FactoryLink, for example, you can have two pens showing data simultaneously. FactoryLink's Delta T feature allows you to associate an offset in time for one of the pens. You could find this feature useful in conducting a comparison between spans of time on a pen. This feature allows you to shift and line up one span of time over another to conduct such a comparison.

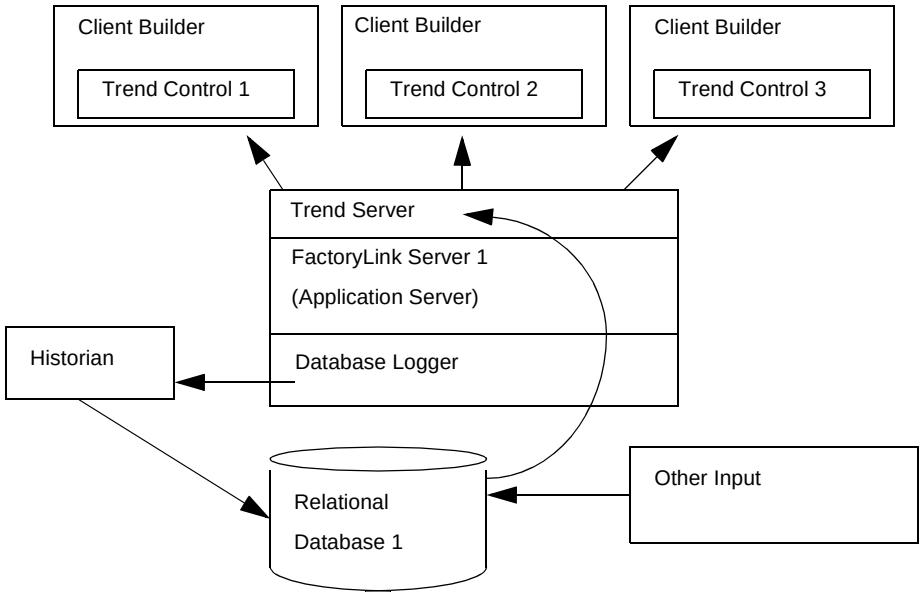
Custom Programming Capabilities

FactoryLink provides everything that you need to construct trend charts that suit most applications. For additional flexibility and customization, FactoryLink also allows custom programming capabilities for the Historical and Real-time Trend Control. You can write a custom program to access the properties, methods, and events included with FactoryLink. For more information about customizing the trend controls, see the *Client Builder Help*.

Trending Components

Figure 23-3 shows a diagrammatic view of the trending components and how they interact.

Figure 23-3 Trending Component Interaction



Trend Server

Trend Server is a program that provides a service to client programs, such as the Historical and Real-time Trend Control. The Trend Server can query any relational database, or many databases, simultaneously.

Relational Database

All trend data configured in the Historical and Real-time Trend Control is stored in a relational database. Data can also come from sources other than FactoryLink's Real-Time Data Base (RTDB).

Trend Controls

The Historical and Real-time Trend Control requests data from the Trend Server. The Trend Server sends the requested data back to the control, which displays the information on the trend chart.

The Historical and Real-time Trend Control is an Active X Control that is a client of the Trend Server. Its container is Client Builder.

The Real-time Trend Control only accesses real-time data from the FactoryLink real-time database and does not interact with the Trend Server or relational database.

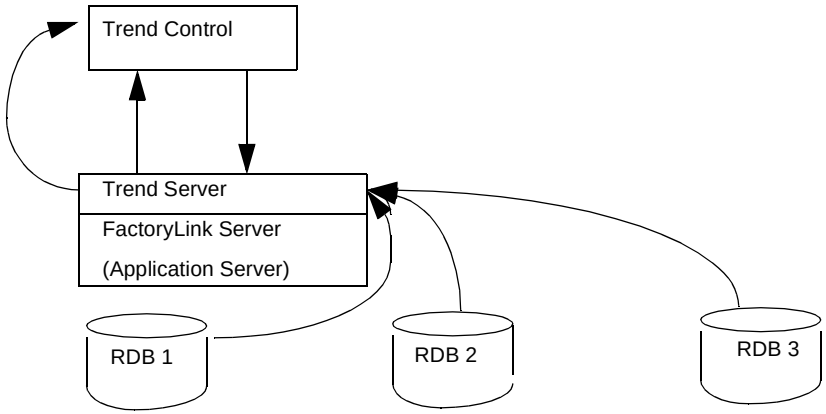
Trend Component Interaction

Since a client can make only one connection, clients can connect only to one Trend Server. Trend Servers, however, can serve multiple clients and connect to multiple relational databases.

A Trend Server can establish multiple database connections as shown in Figure 23-4. Trend Server can establish multiple database connections because the pens that appear on the Trend chart may come from more than one data source. A Trend Server establishes as many connections as needed to get the data as needed by the pens.

Figure 23-4 illustrates the Trend Server query. Trend Control contacts Trend Server and passes the data source information to the Trend Server. Trend Control passes this data to the Trend Server as pens are added to the chart. Trend Server returns the data. Trend Control takes that data and associates it with a pen. This interaction allows a pen to be modified at run time as well as design time. Trend Server gets the data from the relational database (RDB), and sends it back to the Trend Control. This data appears on the Trend chart.

Figure 23-4 Trend Server Query



The FactoryLink Trend Server can access trend data collected by FactoryLink or another data source.The FactoryLink Trend Server component supports an ODBC Data Source along with native connectivity to Oracle, SQL Server, and Access 2000 database. The Trend Server loads the configuration file at startup and connects to the database using the data source name (DSN). The DSN stores information about how to connect to an indicated data provider. The Trend Server reads Trend data from the database, and updates the Trend viewer in Client Builder.

FactoryLink provides flexibility in choosing where to run Trend Server. The recommended place to run Trend Server is on the same node as the FactoryLink Server application, but it can be run on any node. You can choose to put a Trend Server on each client node, or on the node where the FactoryLink Server resides. If you choose to put Trend Server where the FactoryLink Server resides, point the client nodes to this location.

Trend Cluster

A Trend Cluster is a grouping of Trend Servers and their data sources. When you define a Trend Server, you also define a Trend Cluster. You can define multiple Servers.

Figure 23-5 shows a Trend Cluster. A switch will be made to connect to a secondary server if the primary connection fails. For more information about configuring trend clusters, see the *Client Builder Help*.

Figure 23-5 Trend Cluster

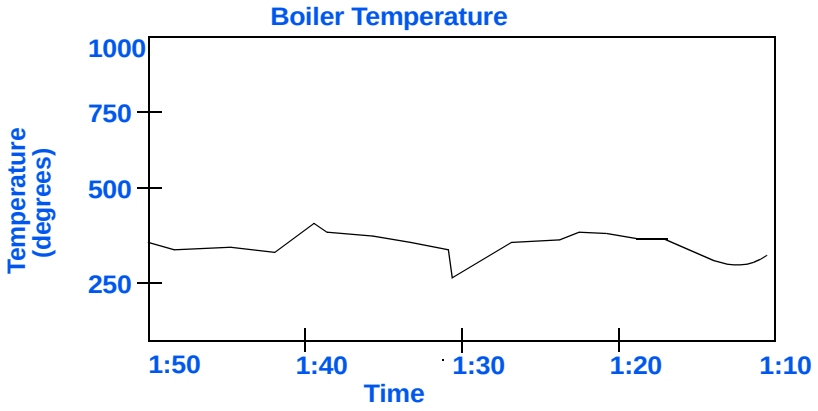
CHART TYPES

Trend charts can be based on time or events.

Time-Based Charts

Time-based charts are best suited for continuous types of data. Figure 23-6 shows a representation of a time-based chart showing a boiler temperature over time. For a time-based chart, the key column in the database is set up as a time field.

Figure 23-6 Time-Based Trend Chart Showing Boiler Temperature



This Trend chart is set at ten minute intervals back through time. Chart direction can start on the right and flow to the left, or start on the left and flow to the right.

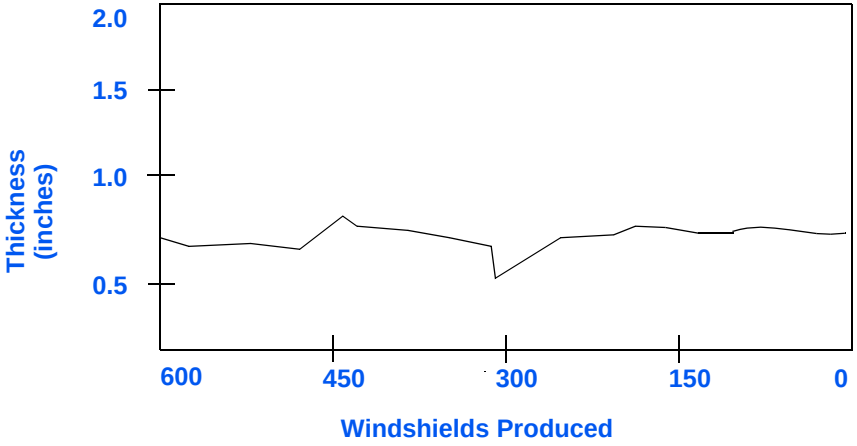
Event-Based Charts

Event-based charts are well-suited for per piece or batch data. For an event-based chart, the key column is set up as a sequence or an ID field.

Per piece (sample)

Per-piece data is data collected for every item in a process. For example, a manufacturer of car windshields inspects the thickness of every completed windshield as it comes off the assembly line. Using an event-based chart, this manufacturer can graphically represent the thicknesses of the windshields produced, regardless of time. Figure 23-7 illustrates an example of an event-based chart.

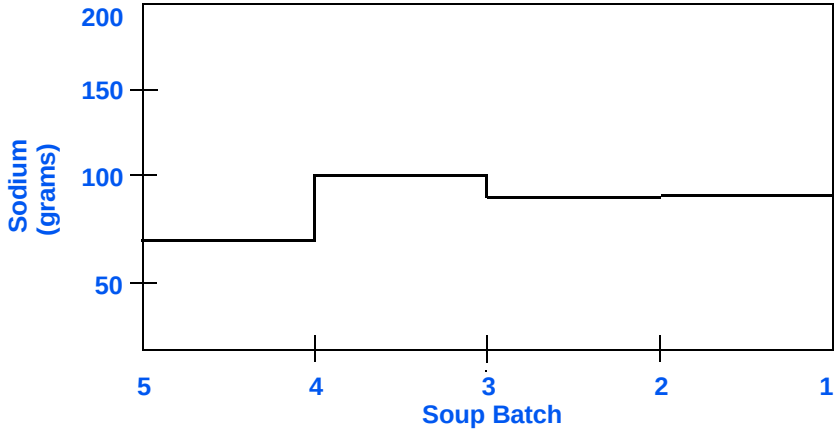
Figure 23-7 Event-Based Trend Chart Showing Per Piece Type of Data



Batch (group)

Group data is data that logically belongs together and is categorized or grouped in that manner. For example, a soup manufacturer that makes two flavors of soup may want to track different batches of both flavors. Using an event-based chart with groups (soup flavors), this manufacturer can graphically represent the differences in sodium content for each group by batch. At the end of the batch cycle, a trigger initiates the sampling of the sodium content for the batch. This sample is written to the database and the sequence number increments to prepare for the sampling of the next batch. Figure 23-8 illustrates an example of an event-based chart that shows batch type of data.

Figure 23-8 Event-Based Trend Chart Showing Batch Type of Data



CONFIGURING TRENDING

All of the configuration for the various trending components occurs in Client Builder. Trend consists of three phases: predesign, design, and run time.

During the predesign phase, you set up a data source name (DSN) so that the database table can be linked to a pen in your Trend chart. In addition, you set up a Trend Server and add it to the configuration. At the end of the predesign phase, you set up a Trend cluster. You work through a series of wizards and dialog boxes to prompt you through the predesign phase.

At design-time, you work through the Trend Control property screens to design your Trend chart. The Trend Control property screen contains four tabs: Aspect, Graph, Pens, and Fonts. All of the properties available on these property pages are accessible in the custom programming environment. The Graph and Pen Tabs contain dialog boxes that are invoked by command buttons. You define the pens of your Trend chart from the Pens tab. In the process of defining these pens, you use the Pens Configuration screen to associate each pen with a data source that you configure during the predesign phase of the process.

At run time, you can use all of the functionality of the Trend task that you can use in design time. Additionally, you can perform the panning and zooming functions during run time in the offline mode only.

For more information about configuring trending, see the *Client Builder Help*.

PROGRAM ARGUMENTS

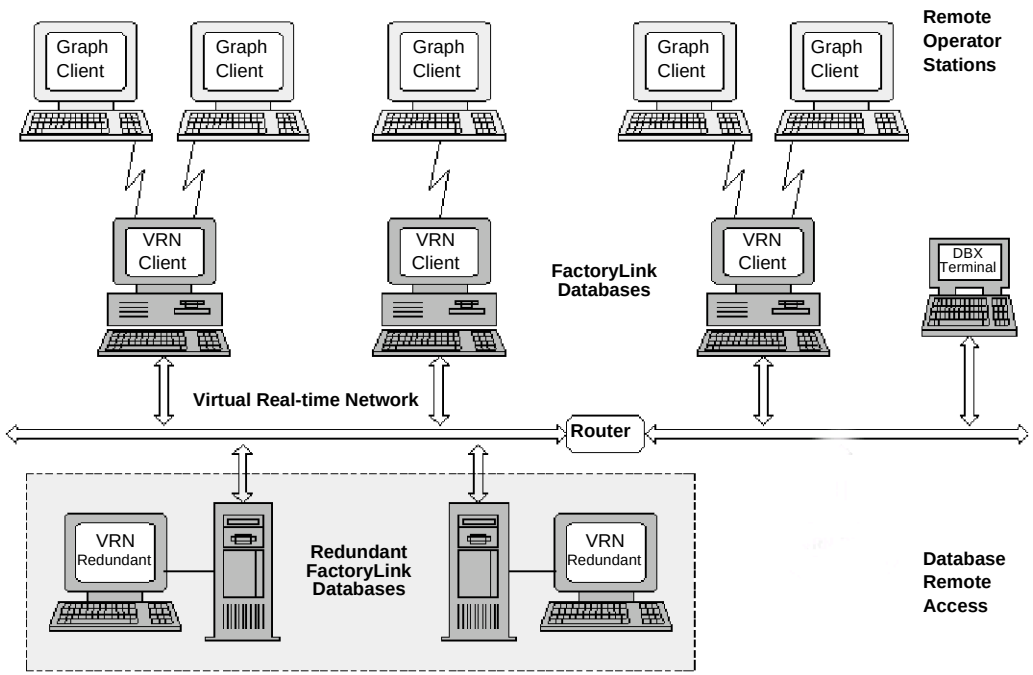
Argument	Description
`-V# or -v#`	Writes trend chart events to a log file.
`-W# or -w#`	Sets the maximum time-out in seconds to wait for a response from the historian. The default is 30 seconds.

- **23** | **TRENDING**
- *Program Arguments*
-
-

Chapter 24. . . .

Virtual Real-Time Network and Redundancy

This chapter contains detailed information for configuring the Virtual Real-time Network and Redundancy (VRN) task for real-time database redundancy. You will also find an overview of possible solutions for configuring historical database redundancy using VRN and other components. The Virtual Real-time Network and Redundancy (VRN) task communicates tag data across the FactoryLink network. This is the mechanism through which the real-time databases of a redundant system are kept in sync. VRN also manages the redundant system's master/slave negotiation and execution. VRN has the capabilities of FLLAN and PowerNet, but also supports the DBX Data Base (X) Terminal, a powerful tool for on-line testing and debugging with local or remote access through a network.



OPERATING PRINCIPLES

VRN is based on the client/server model by applying a unique method for data handling that provides the highest comfort for the operator at the lowest possible network and CPU load. The method ***Action=Reaction*** is applied to any input/output (I/O) data to provide an instantaneous reaction on the local screen while data is transferred to and from the server in the background. The method allows for proper bidirectional data exchange without complex locking mechanisms. VRN can be set up for ***Redundant Servers*** running identical applications including the Alarm Logger. VRN mirrors data from both client and server. Similarly, data may be changed at several locations in a wide area network.

VRN runs the VRN_INIT program at startup to prepare all new or changed configuration data prior to running. If required, you can start VRN_init with arguments, which are described in detail in “Program Arguments” on page 567.

VRN_INIT uses Microsoft software that is installed with Internet Explorer 5.5 or higher. If this software is not found, VRN_INIT will not run.

Configuring VRN is easy. A server simply requires the appropriate information in the FactoryLink System Configuration and a single line in the Connect Control table. Also, clients may require one more line entry in the Client Object Information table. This is because lists of tag names can be specified by wildcards. VRN can be tuned to optimize its update rate to match the operating environment of the application by specifying parameters for ***RdUpdWr*** in the Connect Control or Client Object Information tables. See the “Configuration Tables” on page 533 for detailed information about how to set these parameters. Also, existing FLLAN and PowerNet tables may be easily translated to become VRN configuration tables.

VRN REDUNDANCY COMPARED TO FAULT-TOLERANT PRODUCTS

The following list describes how VRN redundancy compares to fault-tolerant products:

- VRN is a high-availability clustering product that supports both real-time data and load balancing.
- VRN with a redundant historian database would be the preferred solution for applications that can afford historical data to be unavailable for a few seconds. Fault-tolerant solutions may be preferred for applications that must sustain high-value, high-throughput transactions without pause.
- VRN redundancy is composed of two or more servers, all of which are available to run separate workloads. Fault-tolerant systems are composed of multiple servers that perform at the level of a single server. The additional processing power is used for redundancy and overhead to provide its very fast recovery times.

From a technical point of view, there is a wide range of server redundancy from simply having a second server in stock, through an installed (but stopped) cold-standby server, and down to a fault-tolerant system that has either a ready-to-run or a fully operative, hot-standby server like FactoryLink with VRN.

Second Server	Function	Availability, Use, and Investment Cost
Cold-standby	Waiting to be installed from stock	Considerable down time at low cost
	Fully installed but not running	Short down time at moderate cost
Hot-standby	Waiting for auto takeover at failure	High or highest availability at high cost
	Running and fully operative (VRN)	High availability and usage at moderate cost

From a user point of view, redundancy should minimize down time and loss of data due to a system failure. While a cold-standby system may in many cases be reasonable, it may require trained personnel to reinstate it after a failure. This, together with a likely prolongation of the down time, may cost more than a hot-standby server, including the required software. Therefore, when talking about redundancy for FactoryLink, a hot-standby solution is normally assumed.

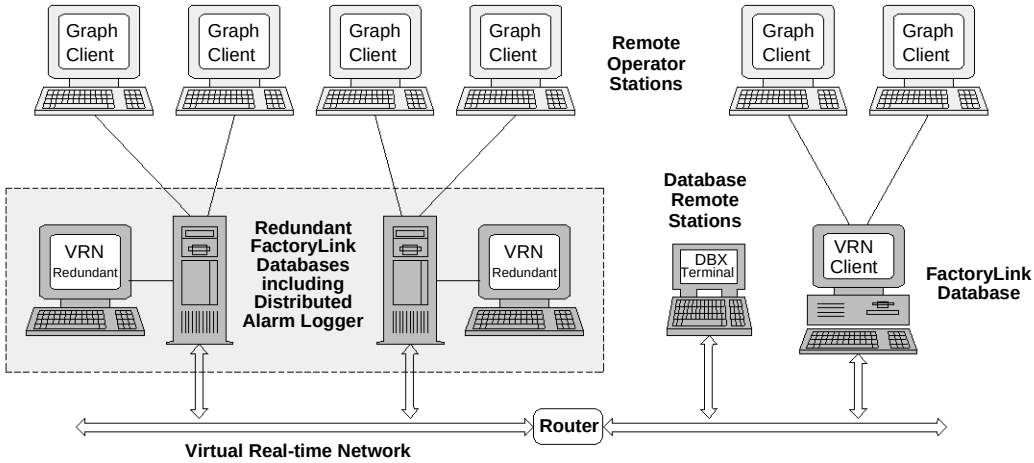
Loss of data normally refers to both real-time and historical information. While historical information can be safeguarded on hard disks using standard node and data management software such as Microsoft Cluster Service (MSCS), real-time data requires special treatment, since it cannot be managed by the operating system. VRN mirrors the FactoryLink real-time database rather than historical files that can be synchronized by standard software, such as SQL Server or Oracle. For historical data, you can run the historians of a redundant system in parallel.

VRN redundancy is not based on just waiting for an auto takeover at failure. At any time all servers involved can be used to their fullest extent. The VRN redundancy cluster supports multiple servers, and VRN clients can automatically reconnect to “1 of x” servers according to a priority level; that is, if a server becomes active, the client automatically reconnects to it.

A VRN redundant system quickly recovers from failures. Except for the Alarm Logger, which is automatically restarted as a server on the actual VRN master, clients would typically not realize a change over. Combining VRN redundancy with a redundant historian database provides both high availability and reliability at moderate costs.

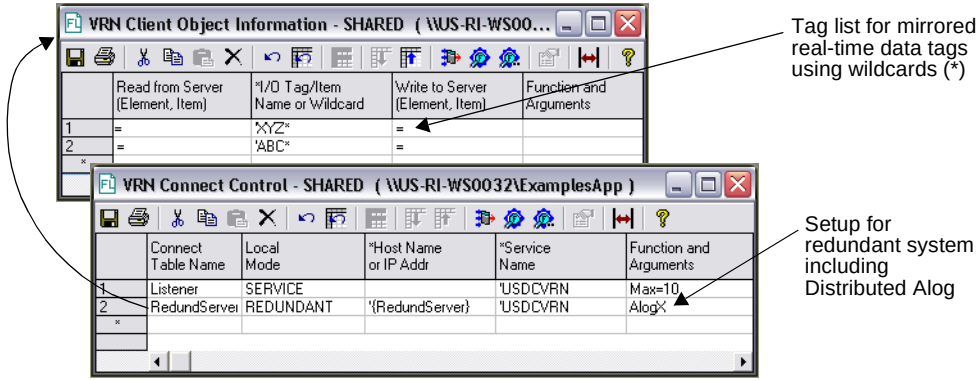
VRN REDUNDANCY MODULE

The VRN module supports redundancy in which identical applications can mirror a selectable subset of real-time database tags. A typical architecture is shown in the following graphic.



Tag selection is done by a simple list that allows for wildcards for quick configuration. VRN automatically controls the Distributed Alarm Logger to run as a client or server on the two redundant FactoryLink stations. Thus, the applications can be kept 100 percent identical.

A typical setup for a redundant partner station is shown in the following graphic. The configuration at the redundant partner station is identical, so you can specify an application, save it, and then restore it on the second computer



RECOMMENDATIONS FOR REDUNDANT APPLICATIONS

The VRN redundancy module is included with a FactoryLink system.You should order a second FactoryLink system for the backup or hot standby server. You need at least one CAL on the backup server so that you can start/stop/monitor the system in a stand-alone configuration.

Driver Recommendations

- Network drivers work best (such as Ethernet, KT, Modbus Plus).
- Serial drivers are very difficult to make redundant without special hardware to manage the serial port communications.
- Need to build application to disable communications on backup server.

Historian Recommendations

- The dBASE IV shipped with FactoryLink is not a good choice for redundancy for trending.
- SQL Server is a much better choice.
- The SQL Server computer should **not** be one of the FactoryLink Servers for a redundant system. Only the primary should log data.
- The SQL Server computer should be running a Server-grade operating system (Windows 2000 Server or Windows 2003 Server) so that the Standard Edition of SQL Server 2000 can be installed. The Standard Edition can use the Replication features to provide data redundancy.

Network Recommendations

- Ethernet networks are preferred with at least 100 MBit network cards.
- Switches, not Hubs, are preferred for networks.
- Dual network cards are preferred for FactoryLink servers with CrossOver Ethernet cable between them (no network switch to fail).

Graphics Recommendations

- CALs in FactoryLink can be combined on redundant servers so that failure of a server allows the full set of clients to attach to the remaining server; there is no need to order twice the number of CALs.
- Client Builder automatically switches between available data servers when configured properly.

- Store the Client Builder project in a shared directory on the SQL Server or other file server on the network to ensure that graphics are available if a server hardware failure occurs.
- Client Builder graphics files can be cached locally.

In a redundant architecture, the clients fail over seamlessly to the available server. Users do not need to click an icon to connect to the backup server. Alarms and PLC data can be synchronized between the redundant servers so that the process is not interrupted and the operator does not need to take any action due to the failover.

Time Synchronization

When running redundant servers, it is important to synchronize the time among the computers.

In Workgroups

For each Windows XP or 2003 computer, open the Windows **Control Panel**, double-click the **Date and Time** component, and click the **Internet Time** tab. If the **Automatically synchronize with an Internet time server** check box is selected, the time is automatically synchronized. If this check box is not selected, select it and click **Update Now**.

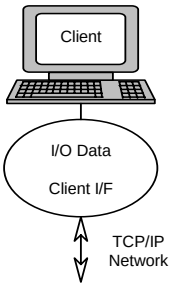
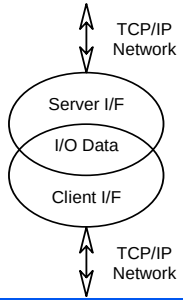
For each Windows 2000 computer, use the following command to synchronize the operating system clock between two computers: **net time \\server_name /set /y**. Alternate software programs are available to perform this function.

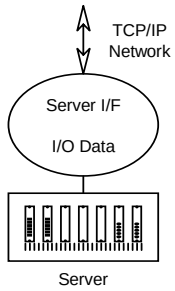
In Domains

The time should be synchronized with the domain controller. If you have questions, contact your system administrator.

CLIENT/SERVER CONCEPT

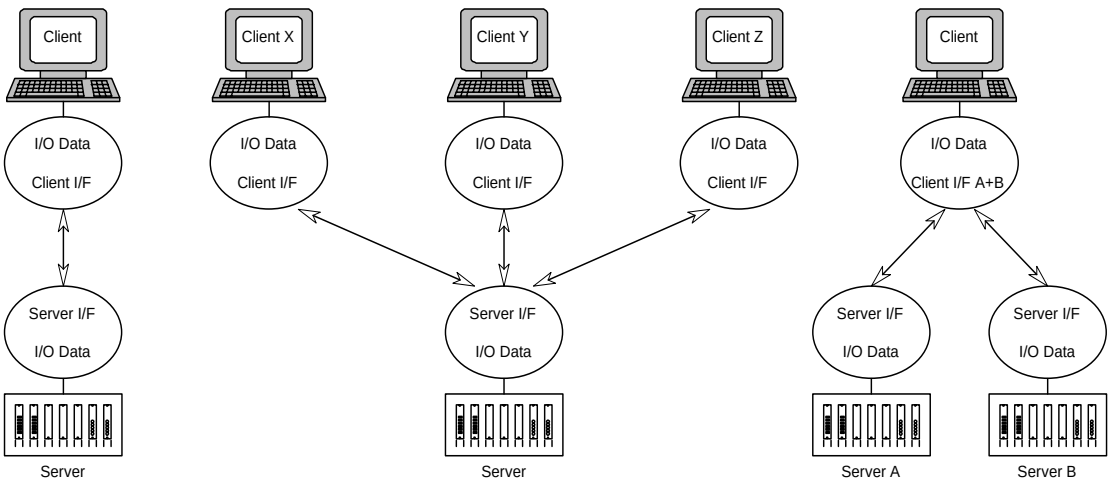
The following table defines the terms used in the text and graphics in this section.

Term	Definition
Client	An application that references input/output (I/O) data from a server (calls for service). It may also send data to a server. Multiple clients of the same kind may appear in a network. A client may be linked to several servers of which I/O data may either be different or multiplexed for redundancy purposes.
Server	An application that sends I/O data to one or more clients (provides service). It may also receive data from a client. A particular server must be unique in a network.
Read / Write	Data from Server to Client is called **Read** and from Client to Server is called **Write**.
	A database containing a process **input/output data image**. Normally, this is the FactoryLink Shared database. However, it may be another data image, such as a driver. Data from the server is mirrored in the client. The system may be compared to Dual Ported Random Access Memory (RAM) as it mirrors data, which may be changed on either the client or server side. Similarly, data may be changed at several locations in a complex network.
	**VRN Client Interface** The Client interface (I/F) supports a local cache for each individual I/O data tag. Individual I/O data is entered to and displayed from the very same database tag, providing instantaneous updates on the local screen while data is transferred to and from the server in the background. The method allows for proper bidirectional data exchange without the need for a complex database-locking mechanism at the server side nor the threat of data consistency problems.
	**VRN Client/Server Node** The local cache allows for both a Client and a Server interface while sharing the memory for I/O data. A Client/Server Node may be used as a repeater to set up a wide area network and, it may be used in a **Redundant System** since the Client/Server roles can be changed on the fly. If applied between server and clients, it provides a chain reaction by updating I/O data of clients as fast as possible on any change caused by any other client or server. Note that any VRN node may play the role of a client and/or server to connect any other VRN node as a server or client, respectively.

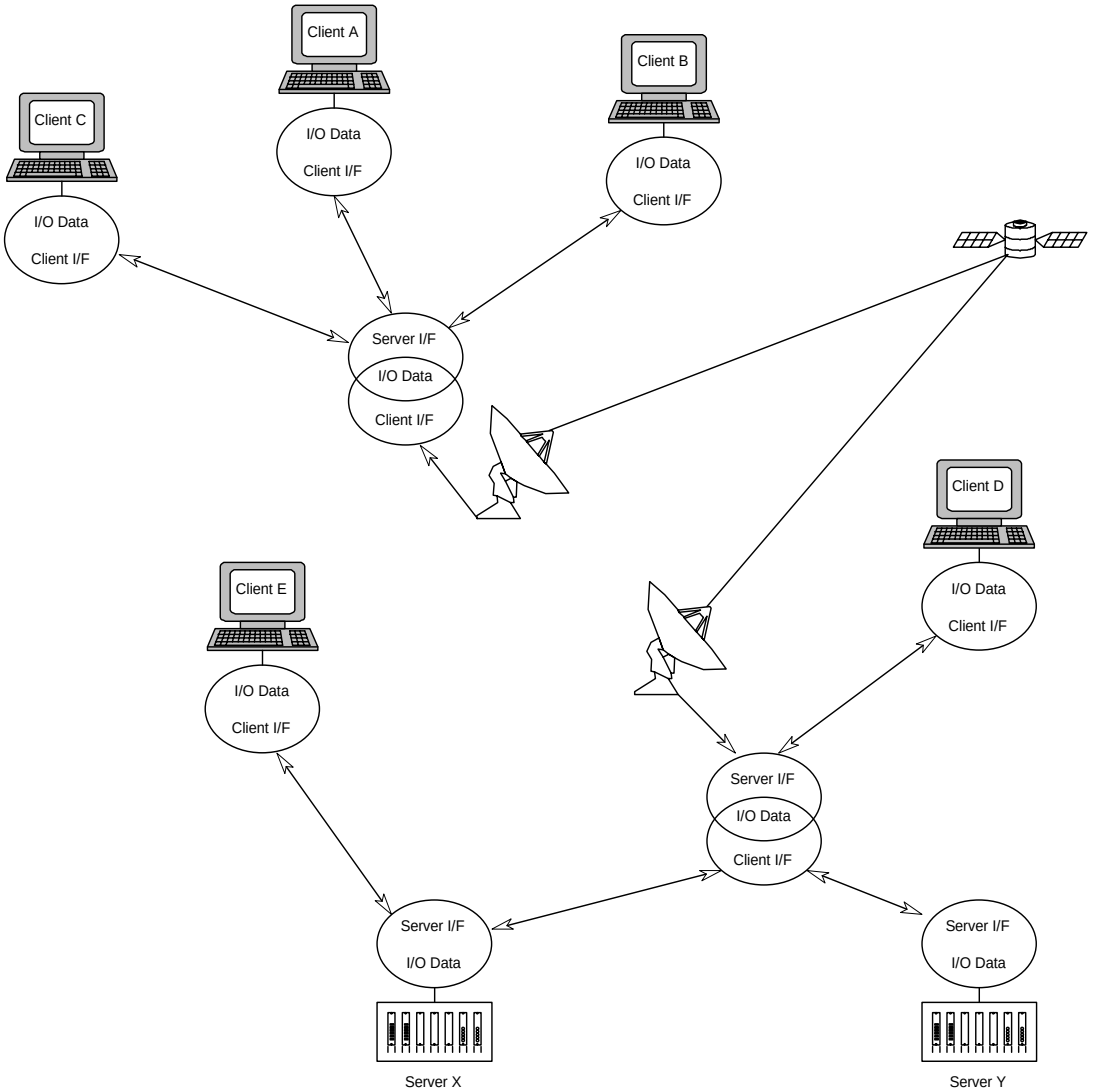
Term	Definition
	**VRN Server Interface** The Server interface supplies I/O data for one or more clients. Regardless of the source, whenever I/O data is changed at the server, it is sent to all connected clients. Because of the local cache being implemented at the client side, it is not imperative to send responses instantaneously by complex interruption and locking mechanisms. Thus, transmission may be kept at normal speed to minimize CPU load and optimize data throughput at the lowest possible network traffic level.

Client/Server Network Structures

Simple VRN Structures in a Local Area Network



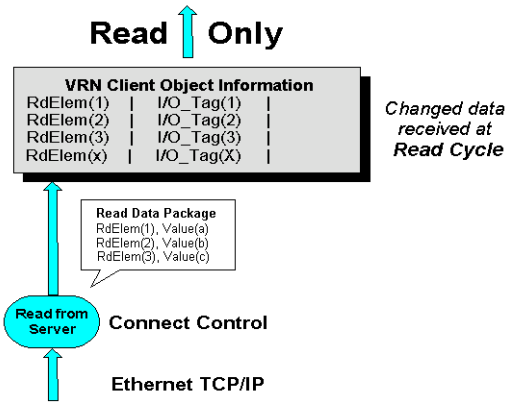
Complex VRN Structure in a Wide Area Network



Client Read and Write Procedures

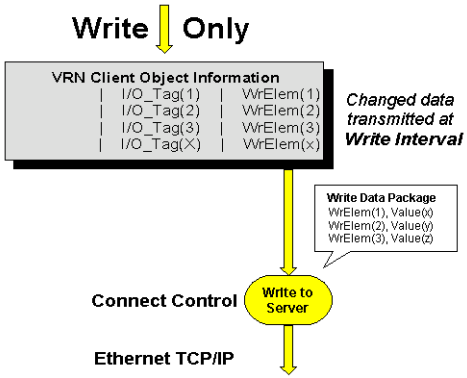
Any I/O Tag/Item may be set up for one-way or bidirectional communication by simple configuration as shown in the following illustrations.

Note: For the following client read and write procedures, use an equal sign (=) if the names in client and server are identical. Use wildcards (???,*) to rename tags or to specify a complete I/O Tag/Item list by a single line entry.



Client Read Data is specified by addressing a *Read from Server Tag or Item* in the Client Object table.

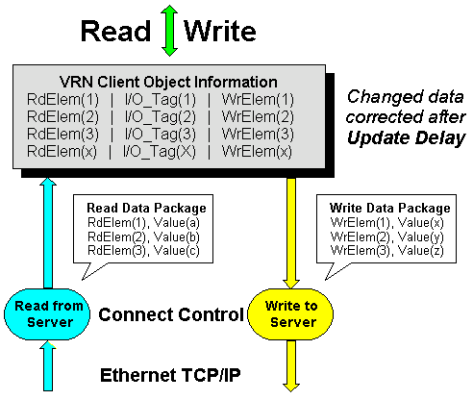
Data changed at the server is sent in packages from there at an adjustable ***Read Cycle*** (event-driven polling).



Client Write Data is specified by addressing a *Write to Server Tag or Item* in the Client Object table.

Data changed at the client is sent in packages from there at an adjustable ***Write Interval*** (event driven transmission).

A possible I/O mailbox is emptied at initialization and/or failure of the link.

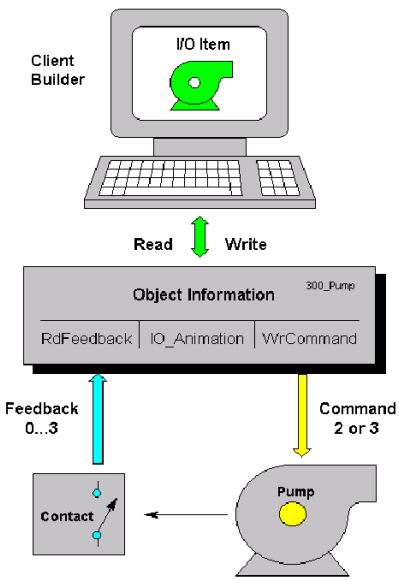


Client Read / Write Data is specified by simply addressing both *Read and Write Tags/Items from/to server* in the Client Object table.

Data changed at the server or client is transmitted while updating at the client side is delayed by an adjustable ***Update Delay***. At initialization, mailboxes are emptied while all other tags are updated from the server before any changes are accepted.

Action=Reaction

Read and Write I/O addresses may or may not be identical. The fact that individual Read/Write data can be combined in a single tag at the client side provides powerful methods for visualization as shown in these examples.

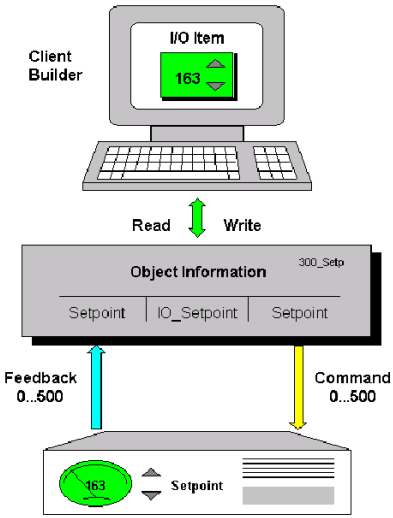


Visualizing a Pump

A motor command sent through the Write channel may be interlocked by hardware and software before it is returned as a contactor feedback signal through the corresponding Read channel. However, the tag in the Client Object Information table may be identical for both command and feedback. Consider a tag, which is used to animate a pump with these values:

0 = OFF, 1 = RUNNING, 2 = STOPPING, 3 = STARTING

To start the pump, set the tag's value to 3 to indicate STARTING by sending the value as a command to the external device. If the feedback signal from the external device indicates a starting pump by a value of 3 in the Update Delay time, the animation remains on STARTING and you have achieved Action=Reaction. Regardless of other delays, the feedback signal may only indicate RUNNING after a while. To stop the pump, enter value 2 to indicate STOPPING. In turn, the feedback will indicate a value of 2 for STOPPING and, after a while return to zero to indicate OFF. Note that all this is done with a single tag at the client side, while control of the pump is possible on either the client or server side.



Visualizing a Setpoint

A setpoint value may be transmitted to and from the same address in the external device emulating Dual Ported RAM through individual Read and Write channels. In this case, it is obvious that the client's animation object is linked with Read and Write tags, which both represent the same setpoint in the external device.

It is not obvious that the setpoint may be changed at any time on either side, client or server. The last action is the one accepted at the external device. If a changed value is not accepted by the external device, the setpoint returns the actual feedback value after the Update Delay. The system automatically takes cares of data consistency by continuously adapting to server data. Due to the Write Interval, data transmission may run at moderate speed still providing fast Action=Reaction for the operator. This unburdens communication even if the setpoint is changed frequently, for example, due to keyboard auto repeating. The setpoint may be automatically pushed to a default or "error"value at startup or failure of the communication. Note that all this is included for each single animation object at the client side.

CONFIGURATION TABLES

The **Connect Control Table** identifies connections, each having its own TCP/IP socket interface. A connection is specified by a local mode for data processing, the partner's host name, and the common services used for the particular link. A local system may play client and/or server on the same machine. For a service, which acts as a listener for possible incoming calls, object information is configured in the partner system(s).

"The **Client Object Information Tables** identify individual object I/Os, which are linked by one or more connects to be read from and written to the server. Read data may or may not be identical to write data. The fact that individual read/write data can be combined in a single object at the client side provides powerful methods for visualization (see also Action=Reaction on previous page and information table above).

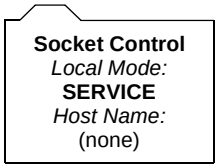
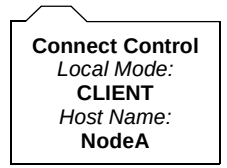
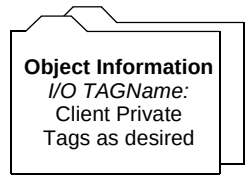
- For example, a start/stop signal WrCommand sent through the write channel by the server to an external device may be acknowledged by a contactor signal RdFeedback received through the corresponding read channel. However, the object I/O Tag/Item IO_Animation in the Client Information table may be identical for both start/stop command and contactor feedback.
- On the other hand, an IO_Setpoint value may be transmitted to and from the very same address in the external device to emulate Dual Ported RAM through communication. In this case, the object in the Client is linked to the same tag for both read setpoint and write setpoint from/to the server side.

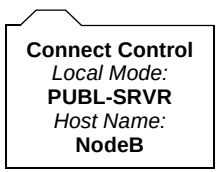
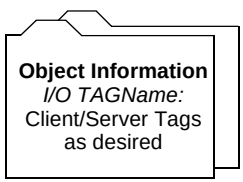
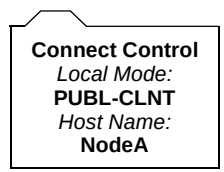
Depending on the application requirements, particular Client Object information may be specified either at the client station or at the server station.

- If specified at the client station, each client may be configured individually by a private list, while the server plays a role as a listener (***SERVICE***), waiting to connect any client (***CLIENT***) upon request. This method may be applied if clients run individual applications.
- If specified at the server station, all clients connected are configured identically by a single public server **(*PUBL-SRVR*)** record, which still uses a listener (***SERVICE***) to connect any public client (***PUBL-CLNT***) upon request. This method may be applied for broadcasting information to multiple clients.

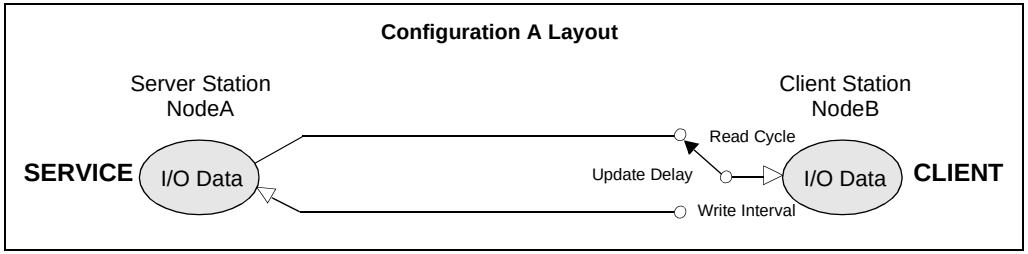
Other modes are configured to set up a redundant (***REDUNDANT***) system or to connect remote nodes by programming.

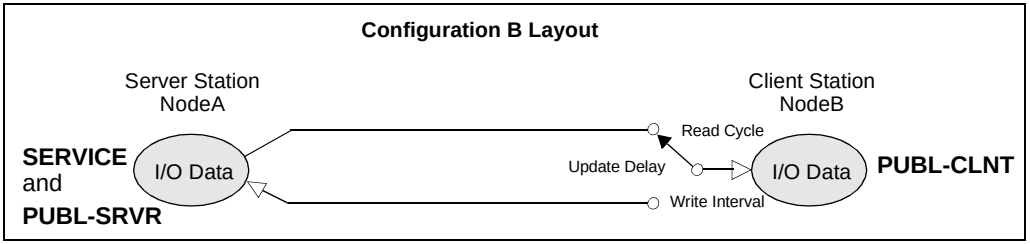
> **Note:** Any VRN node may play the role of a client and/or server to connect any other VRN node as a server or client, respectively. For a redundant system, the roles are even changed dynamically.

Server Station	Configuration Method A	Client Station
 Because a mailbox in the server may fill up if a connection is not established, enter the mailbox(es) to the I/O column of the Client Object table for the service. This will empty the mailbox if the connection is inactive.	← The server is configured for a number of connections (Function *Max*=1..#) by just specifying a listener **SERVICE** at the server station. **Private Client Objects specified at Client Station** Individual Client Object Information tables are specified, referencing the server by a **CLIENT** connection at each client station → See “Emulating FactoryLink PowerNet Read/Write” on page 554 and “Redundant FactoryLink Database” on page 556 to see emulations of PowerNet or FLLAN.	 

Server Station	Configuration Method B	Client Station
 	Public clients can read their Object Information from the server by specifying a **PUBL-CLNT** connection at each client station → **Public Client Objects specified at Server Station** ← A Client Object Information table is specified for multiple public clients by a **PUBL-SRVR** connection at the server station. A listener **SERVICE** is still required to handle incoming client calls.	 Because a mailbox in the server may fill up if a connection is not established, enter the mailbox(es) in the I/O column of the Client Object table for the service. This will empty the mailbox if the connection is inactive.

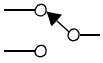
The layouts for configuration method A or B are shown below.





Legend

Indicates the direction of the dataflow.
Note that data transfer from **Server to Client** is called **Read** and from **Client to Server** is called **Write**.



Action=Reaction cache to provide proper bidirectional communication. Data is directed from and to the same process image in the client. This provides instantaneous indication (Action=Reaction) for the client, while data can be transmitted at a low rate in the background using a Write Interval. Data consistency is maintained without the need of a complex database-locking mechanism.

Read Cycle
Update Delay
Write Interval

Transmission control parameters to specify performance and system load. These parameters are set at installation (default) and may be adjusted in the Function/Arguments column of either configuration table.

Connect Control Table

The Connect Control table must be specified for any partner connected to VRN. It identifies the connection names and references for the TCP/IP sockets of both client and server applications. A connection for a service acting as a listener or a server for public information requires a single line entry without a host name. Additionally, a specific or public client requires the host name of the corresponding service to be accessed.

Accessing

In your Configuration Explorer server application, open **Redundancy > VRN Virtual Realtime Network > VRN Connect Control**. The following contains descriptions of the fields in this table, including options that may be selected as values.

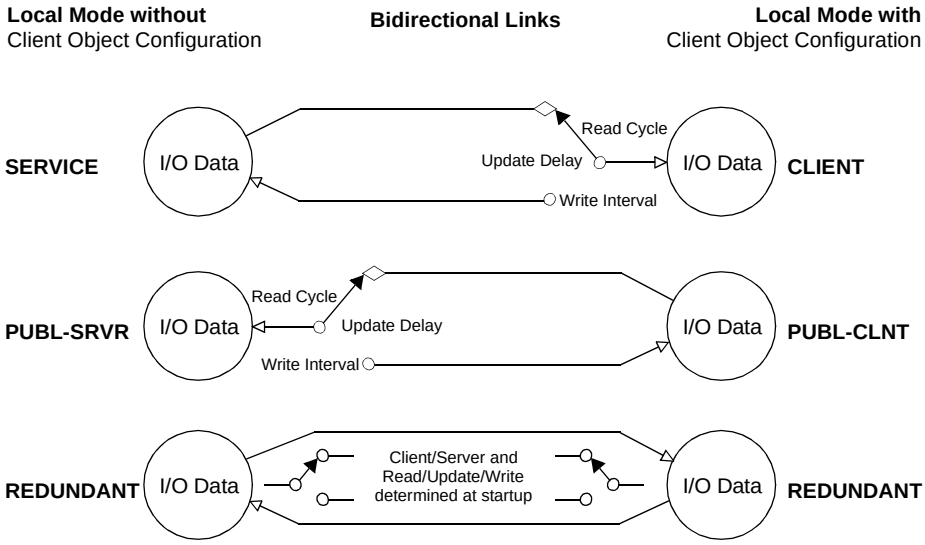
Field Descriptions

Connect Table Name

Name to identify a TCP/IP socket connection, referenced in the Client Object Information table. Valid entries are any text of up to 15 characters. A table name may be entered twice or more in order to link objects to multiple connections using a single Client Object Information table.

Local Mode

Client or server modes are specified for the local or remote system. Local refers to the system where this configuration table resides. Modes include: SERVICE, REDUNDANT, CLIENT, PUBL-CLNT, PUBL-SRVR, PLIB, and dash (-).



SERVICE

This mode defines the TCP/IP service port through which all clients connect, such as REDUNDANT, CLIENT, PUBL_CLNT, and DBX connections. The VRN service port must be assigned a unique name within the operating system's services files. In prior versions, this mode was called DAEMON.

REDUNDANT

A REDUNDANT entry specifies a two-system redundant solution, where one runs as a slave (client) and the other runs as a master (server). Both systems must have both a SERVICE and a REDUNDANT entry configured. In prior versions, this mode was called TANDEM.

Configuration for both master and slave can be identical. You may save an application, restore it on another computer, and run it as a redundant system. The only difference is the system's host file containing the partner's node name or by setting an Environment variable with the partner's computer name. The FLVRNSetup application object in the Examples Application or the FLNEW template creates the {RedundServer} Environment Variable, which must be set to have the partner computer name.

The first system started becomes the master (server), while the second system becomes the slave (client). This is indicated by the corresponding Status and/or Message Tag, and, may be used to control a driver, which only runs on the master. At run time, the slave plays the role of an active client while the master is the server. While being a slave, the system can be used to its fullest extent including all operator functions. Changeover from/to master/slave is done at failure, or you can force the Local Status Tag to become master if set to 1 (odd) or slave if set to 0 (even).

CLIENT

A CLIENT connects to a SERVICE at a remote server station. The data transferred is based on the related configuration located within the local system's VRN Local Client Object Information table. You need to have a CLIENT connection configured to take advantage of mailbox redundancy for drivers. In prior versions, this mode was called PRIV-CLNT.

PUBL-CLNT

A PUBL-CLNT (public client) specifies that the local VRN connects to remote station and transfers tag values as per the VRN Local Client Object Information table defined on the server. Whereas the server defines how the tag data should be transferred, this definition is public and can be shared by any number of VRN public clients. The PUBL-CLNT requires that both a SERVICE and a PUBL-SRVR entry be configured at the remote station. In prior versions, this mode was called PUBLIC.

PUBL-SRVR

A PUBL-SRVR provides the Client Object table for one or multiple PUBL-CLNT connections using this table. It also requires a SERVICE entry be configured at the this station. In prior versions, this mode was called SERVER.

PLIB

A PLIB connection applies the Process Data Image Library (PLIB). Client and server roles are specified according to separate instructions.

-

A dash (-) specifies this connection by the Mode={xx}Function on the same line, here “xx” is an environment variable set to any valid mode. If “xx” is undefined, then this connection is ignored. This entry signifies that the local mode should be determined from the Function and Arguments field.

> **Note:** For backwards compatibility, VRN still accepts the old keywords for the Local Mode: DAEMON, TANDEM, PRIV-CLNT, PUBLIC, and SERVER despite these choices no longer appearing among the configuration choices.

***Host Name or IP Addr**

Constant preceded by a single quote or message tag identifying the host name or IP address of the partner to be connected. Note that IP addresses must be defined within the network, and the host names must be specified in the system host file. You may enter an environment variable using brackets {...} or a tag set by Program Arguments.

For CLIENT or PUBL-CLNT connections, you can enter multiple servers separated by a semicolon, for example: *`NodeX;NodeY;NodeZ*. You can use up to 511 characters. In this case VRN connects to the first node found with a running server. To reconnect the first NodeX in the list (startup default), force the Local Status Tag to zero, the next NodeY=1, NodeZ=2 and so on. If you force the tag to -1, it will select the next available node in the list.

Sample Entry: 199.123.251.2; nodea

Note that for Windows-based systems, a host file entry is optional. However, it provides a faster connection at initialization. No host name is required for a ***SERVICE*** or a ***PUBL-SRVR*** connection. If a Tag contains no valid information, the connection will be disabled. This may be used to disconnect/reconnect a partner. Selection without disconnection is performed by using the *Mux=##* function and argument.

For a SERVICE (listener), a list of one or more servers can be entered to limit which nodes may connect to VRN. If no entry is made into this field, any node may connect.

If the 'ExclHosts' argument is entered into the 'Functions and Arguments' field, the list specifies which nodes may NOT connect to VRN.

Not applicable for PUBL-SRVR or PLIB local modes.

***Service Name**

Constant (preceded by a single quote) or Message Tag (read at VRN startup only) for SERVICE, CLIENT, PUBL-CLNT and REDUNDANT connects to identify the TCP service to be applied or to enter a TCP port number instead. A port number must be specified in all systems involved while only TCP services are allowed. The service name must be specified in the system service file.

Default entry: USDCVRN | 7579/tcp | Tecnomatix

Note that multiple ***SERVICES*** must be set up with individual service names. The default entry "USDCVRN" is set up at installation. An invalid name or tag deactivates the link.

Function and Arguments

Name and attribute(s) specifying a control function (multiple functions to be separated by space). The following functions are not case-sensitive and may be applied on the particular connection (default none).

Max=xx (10 default)

Maximum connection for SERVICE only. If you specify a Control, Status, or Message tag, you must set the array dimension to the Max value.

Alive=xx (30 default)

Alive check timeout 1..999 [sec] for CLIENT, PUBL-CLNT, or REDUNDANT connections. It is used to periodically send a message and check the partner.

TCP_NoDelay

Immediate data transmission for SERVICE, CLIENT, PUBL-CLNT, or REDUNDANT connections. This is achieved by disabling the "TCP/IP Nagle Algorithm" which involves buffering of data until a full-size packet can be sent. **Caution**! Do not use this function unless the possible impact to the network is well understood and desired in order to get the fastest possible data exchange.

Mode={xx}

The Local Mode must be a "dash" for this connection. In this case it is specified by environment variable "xx" which can be set to SERVICE, CLIENT, PUBL-SRVR, PUBL-CLNT, REDUNDANT or PLIB. If "xx" is blank, the connection is ignored.

RdUpdWr=xx,yy,zz,f,n (default 10,30,10)

Transmission control parameters in [0.1sec] for CLIENT, PUBL-SRVR or REDUNDANT connections where:

xx=RdCycle => Poll rate for server data to client

yy=UpdDelay => Time to refresh changed client data

zz=WrInterv => Interval for client data to server

f="forced write at initialization" (option)

n="no data cache" option for xCache SERVER connections

HostPrio=ON (default off)

Enable host priority control for CLIENT or PUBL-CLNT connections only. If set ON, the first host specified in a list of Host Names or IP Addresses will be reconnected when ever available. This can be used for load balancing by referencing a particular host for each client.

Mux=##

Multiplexer for CLIENT or PUBL-SRVR connections only: the link will be enabled if value ## matches the client's Local Control Tag value ##=0..999 else stop transmission (standby). The ## may be compared to be equal [=], less [<], grater [>] or not equal [<>] than the tags value. Note that values ##=1000..9999 are reserved for the Process Data Image Library Plib. No Mux entry (default) enables the link.

Local

A Client Information table for a PUBL-SRVR connection can be specified as a list of local variables that are not exchanged through VRN. This is used to define a pool of variables for the OPC Transaction Cache (xCache) that is used for local data exchange only. (xCache is a VRN client that supports an OPC server interface for local OPC clients such as Client Builder). Note that “Local” specifies the transmission control parameters to RdUpdWr=0,0,0.

Alog[S/X][T[##]]

Valid for REDUNDANT connections only: Distributed Alarm Logger AL_LOG is automatically controlled to run a master on the VRN server (originating the alarms) and as a slave on the VRN client (displaying the alarms).

Suffix S removes the Alarm Status Tag for the slave AL_LOG. Use this flag if Status tags are transmitted through VRN from the server.

Suffix X removes the entire Alarm Definition Information of in the client i.e. the server only uses these tables. For all other options or for a REDUNDANT setup both, client and server Alarm Definition tables must be set identical.

Suffix T adds the Alarm Tag Name to the Message Text field while option ##=1..48 specifies a fixed number of char used for the Tag Name.

ExclHosts

Valid for SERVICE local mode only. This specifies that the 'Host Name or IP Addr' field is to be used as a list of nodes that may NOT connect to VRN, as opposed to the default where it lists what nodes may connect.

Local Control Tag

For a SERVICE connection, you can specify an array of Mailboxes that are used to store feedback information of mailbox requests. The array dimension defines the maximum number of possible concurrent mailbox links. Note that mailboxes using this feature must be marked by Function MbxFb in the Client Information table.

For a REDUNDANT connection: Tag type is Digital. This tag's value corresponds to the local machine's redundant state, where 0 means the local machine is the master and 1 means the local machine is the slave. The tag can be used as a trigger to change the local role by writing the appropriate value (0 for master, 1 for slave).

For CLIENT or PUBL-CLNT connections: Tag type is Digital, Analog, or LongAna to enable the link if the tag value corresponds to Mux=##, or else the transmission is stopped while the link remains active (standby). If the tag is forced written to ##, it can be used as a trigger to update (poll) the corresponding table. Note that no entry (default) enables the link always.

Local Status Tag

Tag of type Digital, Analog, or LongAna to report the current state or error as a value of nibbles N2 - N0:

N2=>Mode: 0 = SERVICE, 1 = PUBL-CLNT, 2 = CLIENT, 3 = REDUNDANT, 4 = PLIB

N1=>Run: 0 = Off, 1 = Running, 2 = Connecting, 3 = Init, 4 - 7 = Error

N0=>Ctrl: Bit0 = Enable/Disable, Bit1 = inverse value

Redundant Mode	Status Tag	
	Analog	Digital
Master (Slave)	785	ON
Slave (Client)	786	OFF
Tries to reconnect / Stand-alone Master	801	ON
Slave initializing	816	OFF

* For more information, see the label messages on page 571.

For a REDUNDANT link, the Status Tag can be used to force the system to become Server (master) if set ON or odd or, to become Client (slave) if set OFF or even.

If multiple servers are specified for a PUBL-CLNT or a CLIENT connection in the Host Name or IP Addr column, you can reconnect the first node in the list (startup default) by forcing the Status Tag to 0, the next node by forcing the tag to 1, the next by 2, and so on. If you force the tag to -1, the next available node in the list is selected.

Local Message Tag

Message to report the current state or error in clear text. The leading hexadecimal number #0x0XYZ indicates the following:

X=>Mode: 0=SERVICE, 1=PUBL-CLNT, 2=CLIENT, 3=REDUNDANT, 4=PLIB

Y=>Run: 0=Off, 1=Running, 2=Connecting, 3=Init, 4 - 7=Error

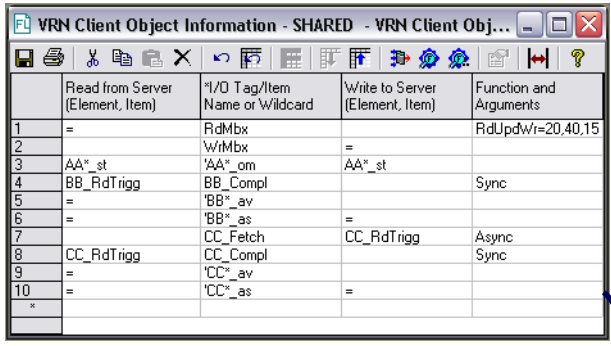
Z=>Ctrl: Bit0=Enable/Disable, Bit1=inverse value

Optional message tag** to report the current state or error of this connection in text format. Message text is defined in file: %flink%\msg\%language%\vrn.txt. For a list of message text, see “Information and Error Messages” on page 571.

> **Note:** If you specify a Status or Message Tag for a ***SERVICE***, you must enter an array of the dimension specified by *Max=##* in the Function/Arguments column. (Invalid array tags will be ignored to prevent undesirable results.)

Client Object Information Table

This table is used to specify client objects for ***CLIENT***, ***PUBL-SRVR***, or ***REDUNDANT*** connections. It contains a list of tags transmitted from and to the server. Wildcards may apply question marks (?) and/or asterisks (*) or embed environment variables using brackets {...}. Thus, data can be addressed with little configuration work or applications can be kept identical. An equal sign (=) in the Read or Write column identifies a server tag that is identical to the tag that specifies the server’s Write=Read name:



In this example, the **Read/Update/Write** rates are modified to **20,40,15** [0.1s]

Two mailboxes, **RdMbx** and **WrMbx**, are transmitted, one in either direction.

I/O Tag/Items AA are renamed and linked bidirectionally to corresponding server tags.

I/O Tag/Items BB are transmitted synchronously on change of **BB_RdTrigg** in the server. The trigger is used as (linked to) the read complete tag in the client.

I/O Tag/Items CC are fetched on change of **CC_Fetch** in teh client. The signal is first sent asynchronously to the trigger in the server; then it is used for example BB.

A detailed, automatically created tag list may resemble this example. Note that VRN will process only the Shared tags.

Accessing

In your Configuration Explorer server application, open **Redundancy > VRN Virtual Realtime Network > VRN Connect Control > "your table name" > VRN Client Object Information**.

Field Descriptions

Read from Server (Element, Item)

Element, Item name, alias or address where data is read from server. Valid entries are: Any text incl. (=) to set element name equal I/O TagName and (?) or (*) for wildcards or, environment variables using brackets {...}. Important: An equal sign (=) specifies the servers Read element to be identical to the clients I/O Tag. A mailbox transmitting data from Server to Client should be entered to the I/O column of a dummy local Client Object table in the server in order to prevent a possible overflow at loss of connection.

***I/O Tag/Item Name or Wildcard**

Name of bidirectional data element or alias where client I/O data is read from and/or written to. Valid entries are Tag name of any type or Wildcard text preceded by a single quote. Wildcards must match the entries of the Read and/or Write column and may apply multiple question marks (?) to replace characters and/or asterisks (*) for strings or, environment variables using brackets {...}. Note that a single asterisk means "all Shared tags excluding mailboxes". Mailboxes are not allowed for bi-directional communication.

Write to Server (Element, Item)

Element, Item name, alias or address where data is written to server. Valid entries are: Any text incl. (=) and (?) or (*) for wildcards or, environment variables using brackets {...}. Important: An equal sign (=) specifies the servers Write Element or Item name to be identical to the Read Element or Item name or to the clients I/O Tag Name if no Read Element or Item has been specified (Write only).

Function and Arguments

Parameters to be applied on the table or on a particular entry (default none): *Default=Tag*; *MbxFb*; *NewList*; *RdUpdWr=##,##,##*,f,n; *Mux=##*; *Exclude*; *Include*; *ExclGlobal*; *InclGlobal*; *Local*.

Default=Tag

Start or default tag value set to I/O tag if a link is not established or is faulty. Note that the Default Function is only accepted for tables using the CLIENT mode. A value is not set if the link is disabled, for example, by the Mux function.

MbxFb

Specifies a mailbox that calls for a feedback that is returned to a hidden mailbox of the sender (for example, DALOGACKMBX for Alarm Viewer). Note that the Local Control Tag of the receiving SERVICE must identify a mailbox array that can be used as a feedback buffer.

NewList

Entry to create a new list as from that line entry. Note that the transmission control parameters (RdUpdWr=) remain the same as specified for the list above. Entry to create a new list from that line entry for the same connection. This may be used to specify a single connection with multiple tables.

RdUpdWr=

##,##,##,f,n

Client/server transmission control parameters in [0.1sec] where:

xx=RdCycle => poll rate for server data to client

yy=UpdDelay => time to refresh changed client data

zz=WrInterv => interval for client data to server

f=forced write at initialization (option)

n=no data cache option for xCache SERVER connections

Specifies the transmission control parameter from that line entry for the same connection with a new *Read Cycle, Update Delay and Write Interval* time described below, *##=0..9999* [0.1s]. Default values are *10,30,10* [0.1s] for items.

- *Read Cycle*–Poll rate to periodically refresh the process image in the client if data has been changed in the server.
- *Update Delay*–time to refresh data of the client if it was changed there. The value may not be less than twice the Program Argument *SleepTime*. If a Write Interval is applied, the delay should be greater than the interval, or it may be set to zero to check the time to receive a feedback.
- *Write Interval*–time for client data output to collect multiple jobs for optimization. Consecutively changed data is stored until the output buffer is full or the interval time has elapsed. Thus, frequently changed data will not overload communication (for example, caused by keyboard repeating).
- Option “f” stands for *forced write at initialization*; that is, all client I/O tags of this list will be set with change flag=ON at first connection or at reconnection (regardless of its status at the server). Note that this also applies for lists that are reconnected by Function *MUX* or when forcing an update by triggering the Local Control Tag.

Mux=##

Mux<##; Mux>##; Mux<>##; Mux=*

Multiplexer: As from this line, data transmission is only enabled if the value ## matches the client's Local Control Tag value. The ## may be compared to be equal [=], less [<], grater [>] or not equal [<>] than the tags value. Mux=* will enable a list unconditionally as from that line. Note that values ##=1000..9999 are reserved for the Process Data Image Library PLIB.

Exclude / Include

Exclude a list of tags as from this line entry. Excluded variables are not transmitted through VRN, but are removed from the list above in the same table. Function *Include* (default) is used to cancel the *Exclude* function.

ExclGlobal / InclGlobal

Exclude global and system tags as from this line entry from the list of a CLIENT, PUBL-SRVR or a REDUNDANT connection. Excluded global variables are not transmitted. These variables are:

a) All Global tags, for example: A_SEC, DATE, TIME, and so forth.

b) All tags configured in the System Configuration table

c) All tags configured in the VRN Connect Control table

Function *InclGlobal* (default) is used to cancel the *ExclGlobal* function.

Local

Specifies a list of local variables for a PUBL-SRVR connection as from this line entry. Local variables are not transmitted through VRN, but they are automatically excluded from ALL lists within this table. Local variables are specified for xCache (OPC Transaction Cache), a VRN client that supports data exchange between local OPC clients such as Client Builder. Note that “Local” specifies the transmission control parameters to RdUpdWr=0,0,0.

Sample Local Table:

Read from Server (Element,Item)	*I/O Tag/Item Name or Wildcard*	*Write to Server (Element,Item)*	*Function and Arguments*	
=	´*	=	***Local***	← Specifies a list of all Shared tags (bidirectional)
	´*$*			← Excludes all tags with a $ and makes them “Local”

Sync / Async

Specifies synchronous or asynchronous data transmission from that line entry. Asynchronous (default) means that data is sent on change including the change flag. Synchronous means that data is transmitted simultaneously at startup and then at **any change** of the Read and/or I/O Tag/Item entered in line with the *Sync* function. Data comprises the line with *Sync* and includes all lines downwards in the table until a new *Sync* or *Async* function is entered. The ReadTag is the trigger in the server to send Read data, while the I/OTag is the trigger in the client to Write data. Async is always active unless overruled by Sync (Async is used to cancel Sync). Async is used to cancel the Sync function, it is always active unless overruled by Sync.

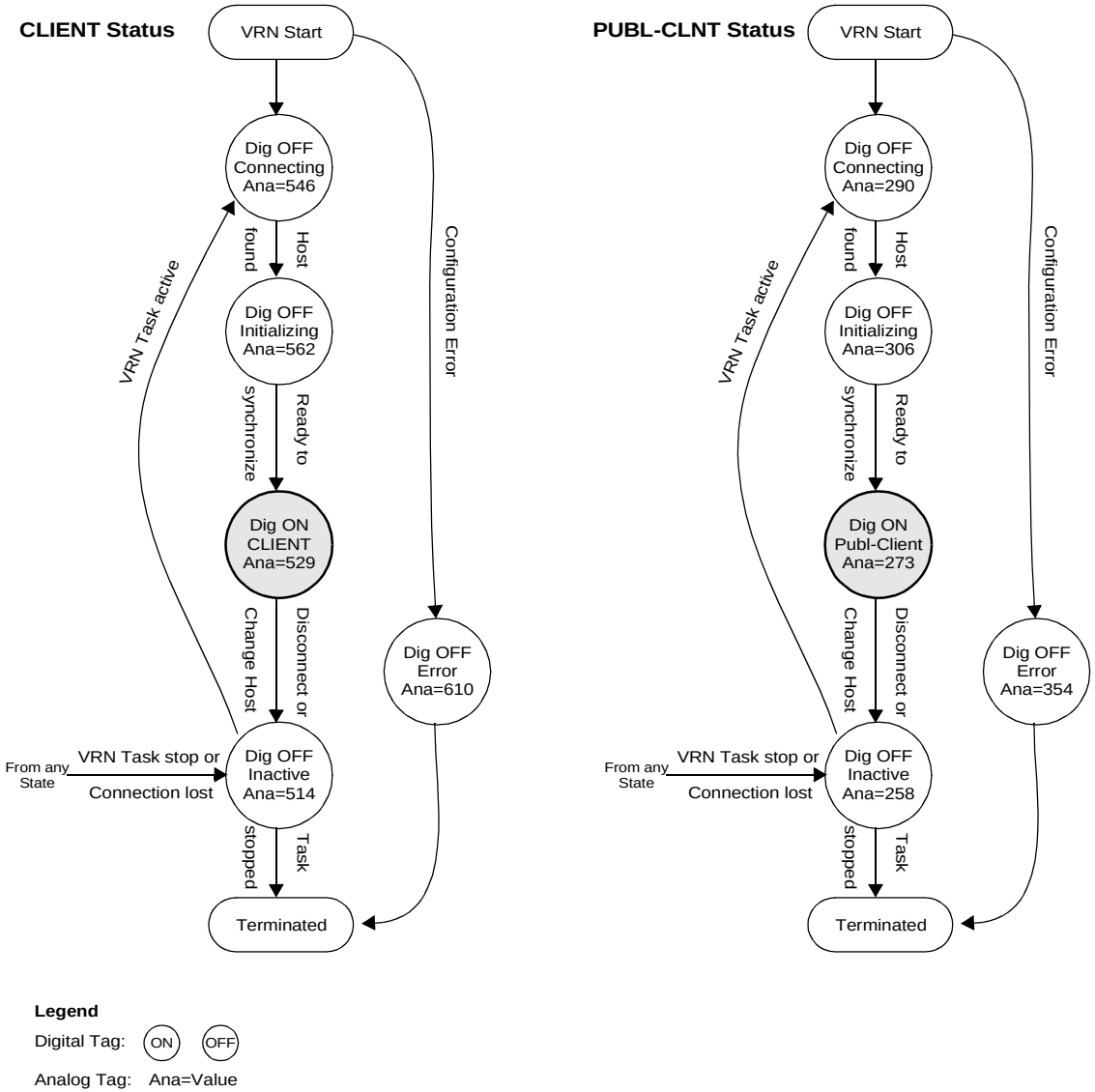
The following Client Object Information table shows all entries accepted for Read, Write and I/O Tag/Item when using function Sync and Async.

Sample Sync/Async Table:	***Read from Server (Element, Item)***	***I/O Tag/Item Name or Wildcard***	***Write to Server (Element, Item)***	***Function and Arguments***	**Transmission Method**		**Rd Trigger in Server**	**Wr Trigger in Client**
Synchronous send:	*(any)*	***I/OTag*** *(any)*	***WriteTag*** *(any)*	***Sync***	(Async Read)	**Sync Write**	(none)	**I/OTag**
Synchronous receive:	***ReadTag*** *(any)*	***I/OTag*** *(any)*	*(any)*	***Sync***	**Sync Read**	(Async Write)	**ReadTag**	(none)
Synchronous send/receive:	***ReadTag*** *(any)*	***I/OTag*** *(any)*	***WriteTag*** *(any)*	***Sync***	**Synch Read+Write**		**ReadTag**	**I/OTag**
Asynchronous send/receive:	*(any)* *(any)*	*(any)* *(any)*	*(any)* *(any)*	***Async***	Asynchron Read+Write		(none)	(none)

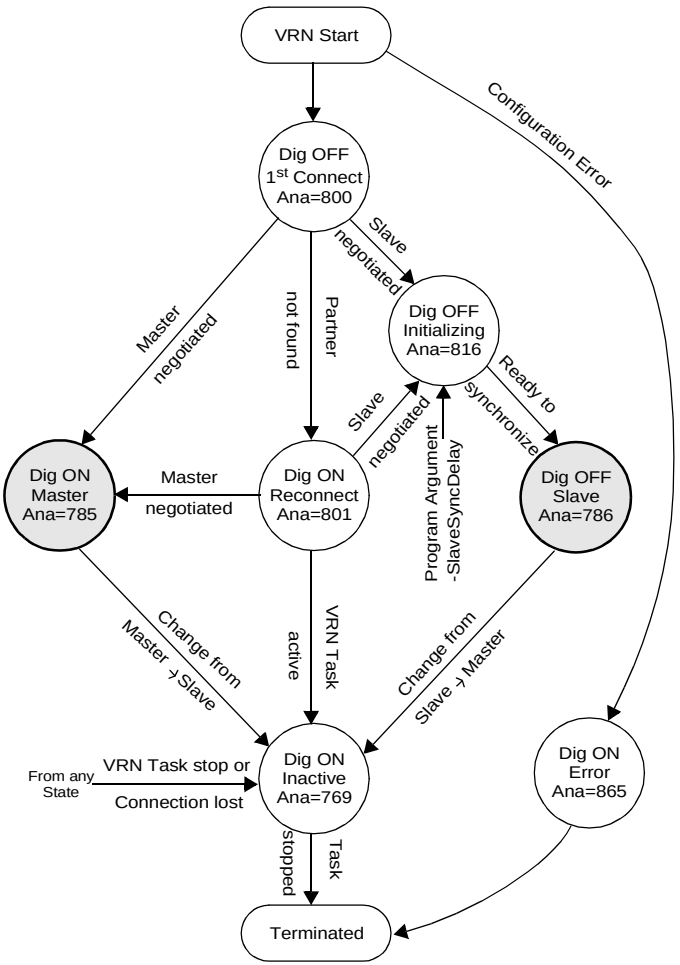
Note: Use **explicit tags** (no wildcards) in the ***Sync* line** (apply dummy entries if a tag is not used). Except for the *Sync* line itself, synchronized data is sent at triggering regardless of data changes. When entering **Mailboxes** in a ***Sync* table**, one only message will be transmitted at a time.

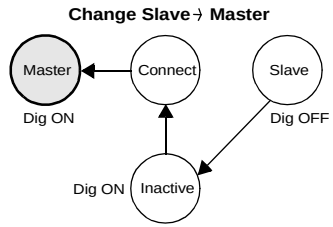
If you want to **Poll** or **Fetch** data from a server, specify a ***Sync* Read**, then write to the ReadTag in the server by a separate Async line entry. Note that the I/OTag of the *Sync* Read is the **Read Complete** trigger. If you wish a **Write Complete** trigger, simply read the WriteTag to an I/OTag by a separate Async line entry.

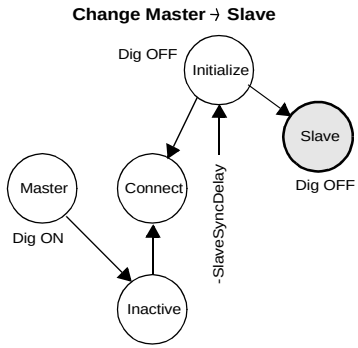
CLIENT, PUBL-CLNT, AND REDUNDANT STATE EVENT DIAGRAM



REDUNDANT Status







Legend

Digital Tag: (ON) (OFF)

Analog Tag: Ana=Value

APPLICATION OBJECTS SIMPLIFY VRN CONFIGURATION

Using the VRN task is easier than it looks if you use the application objects provided with FactoryLink. The Examples Application, FLNEW templates, and Application Setup Wizard all create a server applications that contain an application object that makes setting up the VRN task just a matter of providing the following information:

- Remote node name for redundancy
- Whether or not the application uses ODX with ECI
- Whether or not the application uses a RAPD driver with IOXlator

If you use ODX with ECI or RAPD with IOXlator, the VRN task uses a feature called mailbox redundancy. In a perfect redundant application, the only data that needs to be synchronized between the servers is the I/O data. This means that you should put every driver tag in the VRN tables or use a simple naming convention so that you transmit all PLC tags using the VRN wildcard function. This technique can be problematic because it is too easy to forget a tag. If you are using an ECI or IOXlator supported driver, there is an easier solution.

Mailbox redundancy uses VRN to route mailbox tags either locally and/or across the network to the redundant server. The reason this is the perfect solution is because you don't have to set up any tags other than the 4 standard ECI or IOXlator mailbox tags.

A driver works on a stand-alone computer as follows:

PLC <-protocol-> Driver <-mailbox-> IOXlator <-> tags

In a redundant system the VRN task manages the mailbox tags so that the slave can disable the driver functions but still have the IO synchronized with the master system.

Master: PLC <-protocol-> Driver <-mailbox-> VRN <-mailbox-> IOXlator <-> tags

Slave: \-> VRN <-mailbox-> IOXlator <-mailbox-> tags

The application object uses the tag VRN_CONTROL to control the mailbox redundancy.

Master: VRN_CONTROL=0

Slave: VRN_CONTROL=1

VRN_CONTROL is used in the application to disable various functions like driver communications and database logging. This tag allows the developer to design the application to function correctly in master/slave modes without writing IML to manage the failover.

HISTORICAL DATABASE REDUNDANCY

VRN is used for real-time database redundancy, but it also plays a role in the solutions for historical database redundancy. In historical redundancy, you use VRN to establish the master/slave relationship between two FactoryLink servers that run in tandem and use Microsoft SQL Server 2000 for historical data storage. Historical database redundancy also assumes that FactoryLink runs on a Microsoft Windows 2000 or XP operating system.

FactoryLink supports several methods of handling redundancy of historical data with varying degrees of complexity and cost:

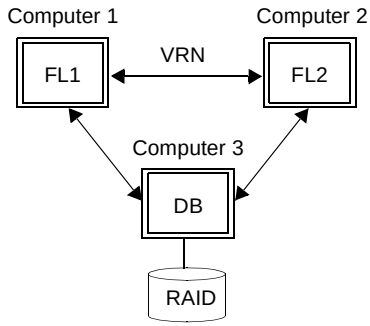
- Single historical database with hardware redundancy
- Snapshot historical database replication
- Transactional historical database replication
- Merge historical database replication
- Clustered historical database replication

For the most of these methods, it is suggested that the backup server logging be disabled by using the **VRN_CONTROL** tag. There is a field in the DBLOG control table for this function. Using this tag will make only the primary computer provide logging functions at one time.

Most of the methods (except single historical database logging) use SQL Server's built-in replication technology. These methods are described in the following sections and are presented in the order of **simplest** to most **complex**.

Single Historical Database with Hardware Redundancy

In a single historical database logging configuration, FactoryLink resides on two computers and the database resides on a single computer with internal or external RAID drives.

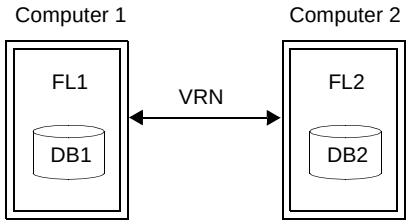


PRO: This configuration is the simplest and easiest to implement and is very reliable.

CON: This configuration has a single point of failure, but it can be improved by using failure-tolerant RAID drives and redundant power supplies.

Snapshot Historical Database Replication

In a snapshot historical database replication configuration, FactoryLink resides on two computers each with a local database. The data from FL1 is saved to DB1 and the data from FL2 is saved to DB2. The data from these two databases is not synchronized automatically. If Computer 1 fails, then Computer 2 continues logging data. Later, when Computer 1 is restored, a hole is left where data was missed. You can then replicate the complete set of data from the Computer 2 database to the Computer 1 database using snapshot replication.



PRO: This configuration is for when you want to capture only historical data, and it is acceptable for the data to reside in multiple databases. The database can be replicated using snapshot replication with no loss of logged data.

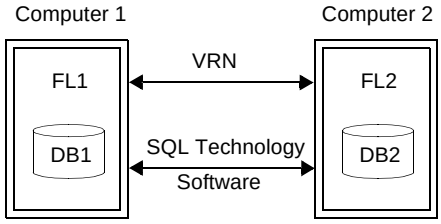
CON: You would need to stop data logging operations while the database is being replicated.

> **Note:** Refer to Microsoft's SQL Server documentation and technical support for details on implementing a snapshot historical database replication solution.

Transactional Historical Database Replication

In this configuration the databases are synchronized. Transactional historical database replication is used when:

- Data must be replicated as it is modified.
- Transactions must be preserved.
- Servers are reliably and/or frequently connected through the network.



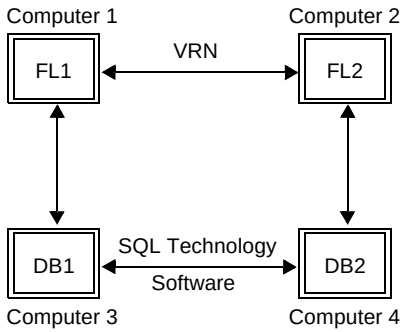
PRO: This configuration uses two computers each with local databases and has no single point of failure. Additional hardware is not required. For most users, this configuration is the most obvious solution. Databases do not need to be stopped to replicate data after a failure.

CON: Data has to be noncritical since during a failover, there will be a time window when a small amount of data might not be captured. After a failback, the saved data must be restored back into the primary database. This configuration requires using SQL Server Standard Edition and has to be implemented by the proper personnel to use the SQL replication technology.

> **Note:** Refer to Microsoft's SQL Server documentation and technical support for details on implementing a transactional historical database replication solution.

Merge Historical Database Replication

Merge historical database replication is used for critical data that cannot be lost. The databases are synchronized and changes are tracked on both databases representing a single version of the data. If the FL1 computer is down, constant database synchronization continues. If the DB1 database is down, FL1 continues logging to DB2; and when DB1 is backed up, DB2 automatically replicates the data to DB1.



PRO: This configuration uses four computers and has no single point of failure. You do not have to stop operations to resynchronize data. Even if an application computer is down, constant database synchronization continues. If the primary database is down, logging continues on the second database.

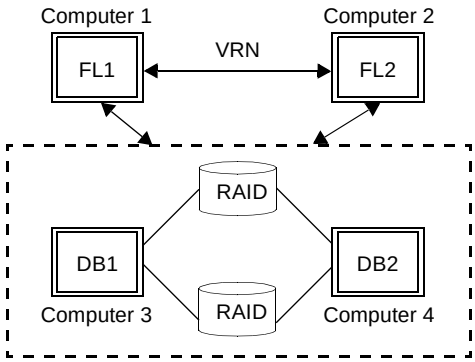
CON: Because this configuration requires more CPU time, there is a higher hardware requirement of four computers. This configuration has to be implemented by the proper personnel to use the SQL replication technology.

> **Note:** Refer to Microsoft's SQL Server documentation and technical support for details on implementing a merge historical database replication solution.

Clustered Historical Database Replication

The most reliable and fault-tolerant method of providing redundancy for an SQL database is to run SQL Server on a clustered Windows server.

For a clustered historical database solution, no special work is required in your FactoryLink application because with a clustered database, FactoryLink sees only a single virtual database, and the redundancy is handled transparently by the clustering.



Virtual Database (Viewed as a Single Computer)

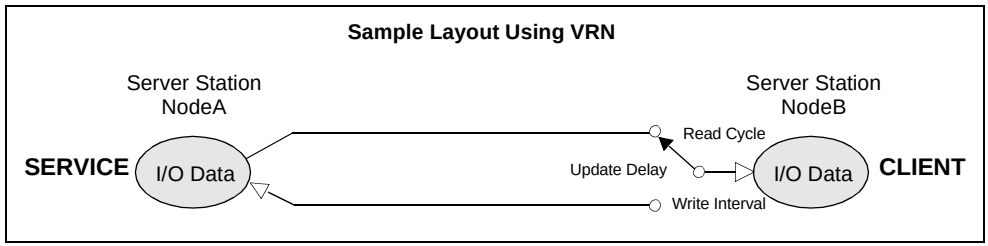
PRO: This configuration provides a fully redundant solution and is the surest database redundancy method for both small and large databases. Clustering is straight forward to implement because of the single virtual database. This is the most fault-tolerant method for replicating data.

CON: This configuration is the most expensive because of additional hardware, software, and implementation costs.

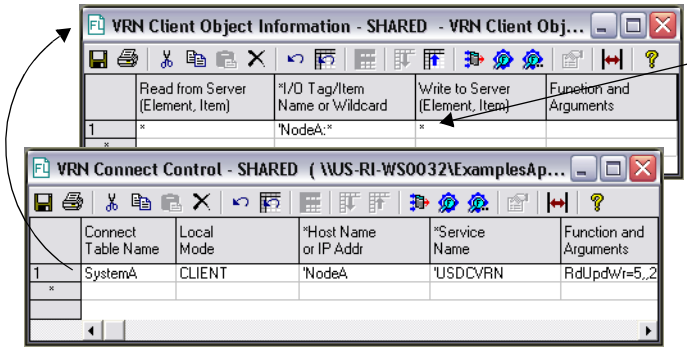
> **Note:** Refer to Microsoft's SQL Server documentation and technical support for details on implementing a clustered historical database replication solution.

EMULATING FACTORYLINK POWERNET READ/WRITE

The configuration below emulates PowerNet in the following sample layout:



Client Object and Connect Configuration at NodeB



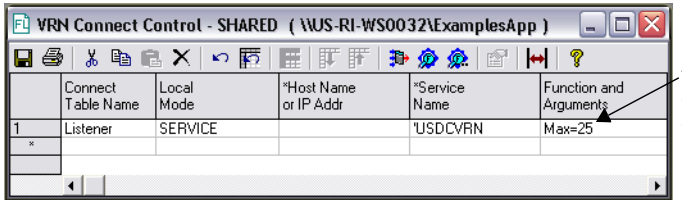
Bidirectional communication for all tags of NodeA

If you desire Read-Only or Write-only, simply remove the corresponding asterisk from the table.

At any time, you may add explicit tags or extend the list by entries shown in other examples.

TCP/IP Network

Server Connect Configuration at NodeA



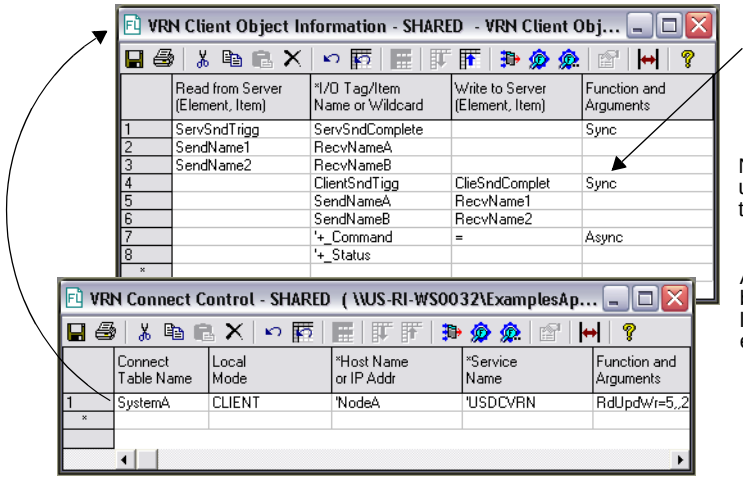
This connect specifies a listener, which can communicate with up to 25 clients simultaneously.

Note that the default transmission control for Read Cycle, Update Delay and Write Interval may be replaced by a parameter entry in the Function/Arguments column in either table. Wildcard entries in the information table are possible and provide a lot of flexibility: you may specify I/O Tag/Items, which are read only, write only, or, you may rename (alias) tags and, you may even enter different tags for read and write at the server side. In addition, the system accepts mailboxes, and an existing PowerNet table may be easily translated to become a VRN configuration table.

EMULATING FLLAN SEND/RECEIVE

The configuration below emulates an FLLAN Send and Receive table for the sample layout shown below. Note that receive and send information is transmitted in one direction only (not bidirectional) and that only one Client Object Information table is required for configuring both client and server side:

Client Object and Connect Configuration at NodeB



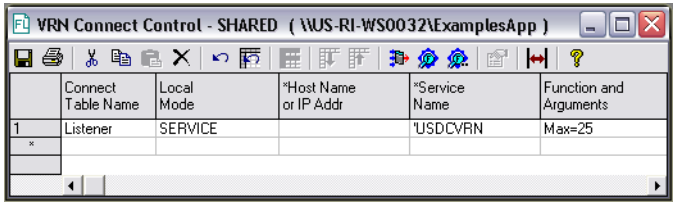
LAN send and receive information using *Sync* (block) and *Async* (exception)

Note that the *Sync* Function is used to specify block send tables.

At any time you may add bidirectional tags or extend the list by entries shown in other examples.

TCP/IP Network

Server Connect Configuration at NodeA



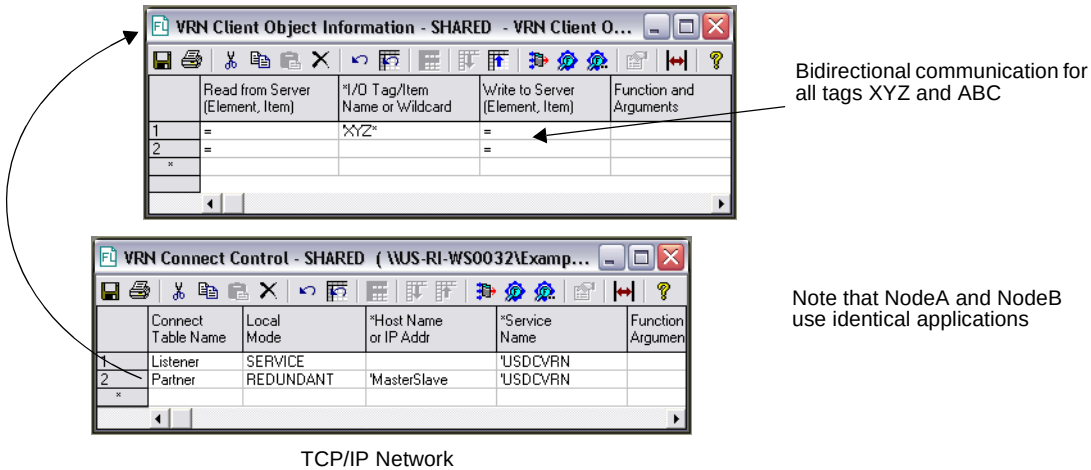
This connect is the same as on the previous example and specifies a listener for up to 25 clients.

Note that network alias tag names are not required due to automatic renaming at the client and server side. The default transmission control for Read Cycle and Write Interval may be modified by a parameter in the Function/Arguments column. You can use wildcard entries to make configuration easier. As shown, an existing FLLAN table may be easily converted to become a VRN configuration table. The Sync Function also allows for configuring complete triggers.

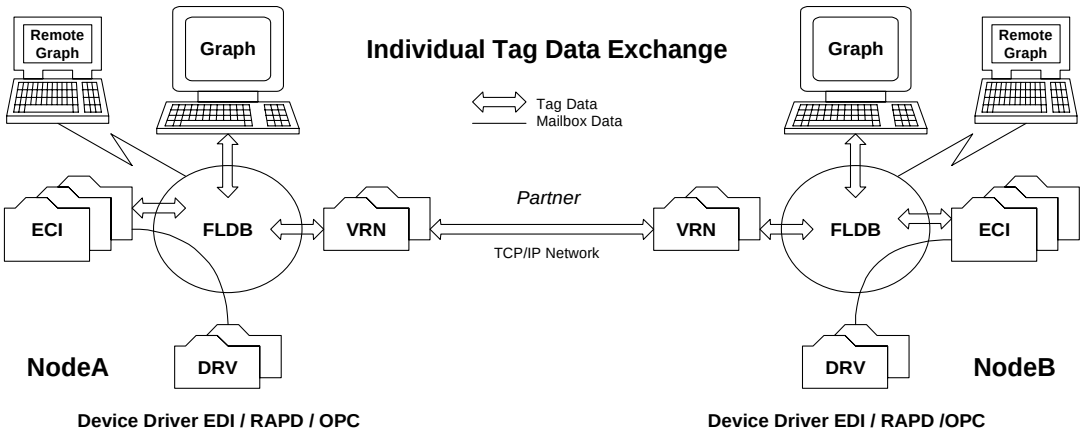
REDUNDANT FACTORYLINK DATABASE

The configuration below uses the Redundant Mode to set up a redundant system with NodeA and NodeB. The first system started will automatically become the master (server), while the second system will be the slave (client). This is indicated by the corresponding Redundant Status and/or control tag and may be used to control a particular device, such as a driver, which may only run on the master system.

VRN Client Object and Connect Configuration at NodeA and NodeB



The Redundant Mode specifies a connection, which either runs as a master or a slave. Configuration may be identical for either node. Thus, you may save an application, restore it on another computer and run it as a redundant system. The only difference is the system's host file containing the partner's node name. Wildcard entries make configuration much easier and are used to specify Tag data exchange. For example, the simple configuration example can be set up as a redundant system setup.



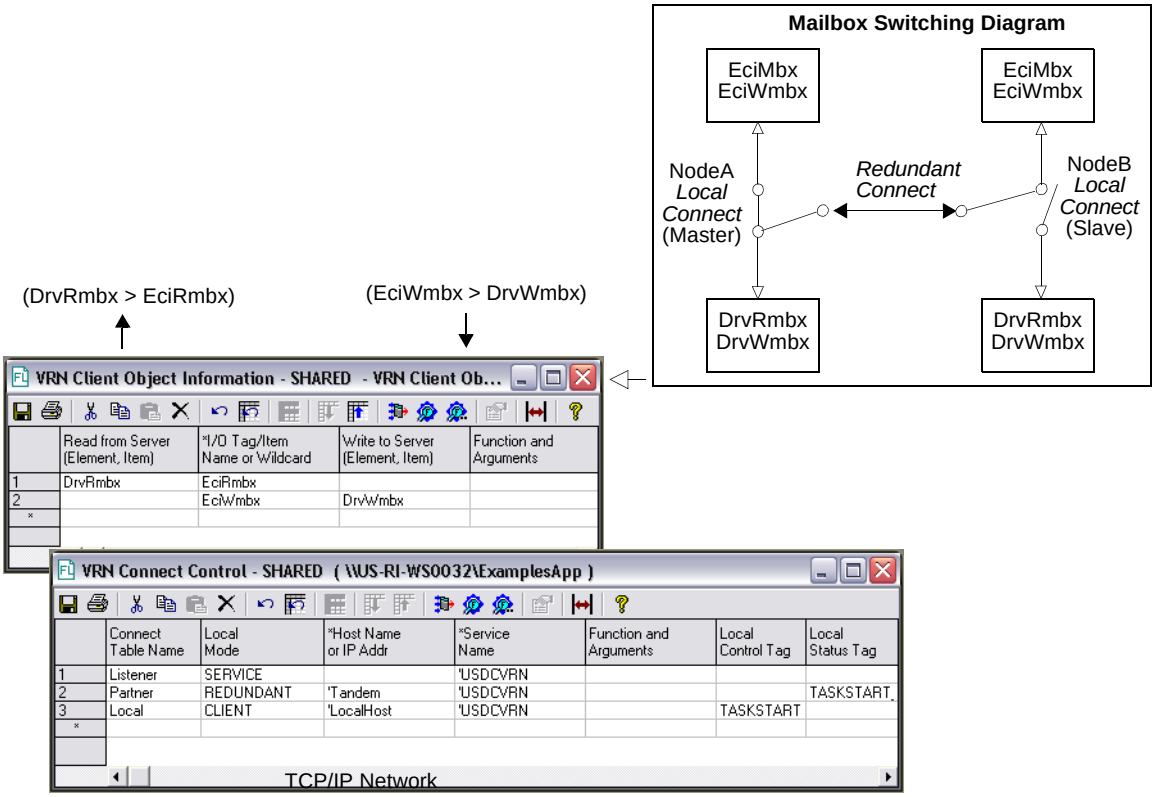
In this example, all Shared tags named XYZ and ABC are automatically exchanged to keep data consistent between the two systems. The **VRN_CONTROL** and **VRN_STATUS** tags can be used to control the driver read/write tables and ECI task (enable/disable). And, you can use this tag to force the system to become master if set to 1 (odd) or to become slave if set to 0 (even).

Note that for ECI with RAPD or OPC Data eXchange (ODX), it is recommended to exchange Mailbox data directly as shown on the next page. This is more powerful as it exchanges binary data between the driver(s).

REDUNDANT RAPD OR OPC DATA EXCHANGE WITH ECI

The configuration below shows the setup for a redundant system transmitting Mailbox data of RAPD or OPC Data eXchange (ODX) from and to ECI Decoder through a Redundant connection. This is powerful for data distribution since ECI datasets contain compact information, which is decoded only at the target system.

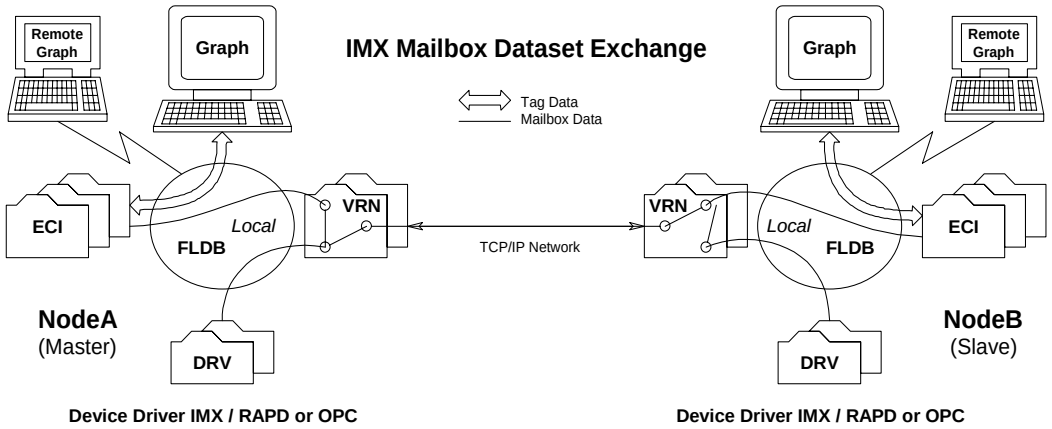
VRN Client Object and Connect Configuration for NodeA and NodeB



Note that the VRN Object Information entries are identical for both Partner/Redundant and Local/CLIENT connection. If you wish to exchange additional data through the Redundant link, simply add the information to the Partner table, but not to the Local table.

> **Note:** When you declare an OPC connection in ODX, you configure the OPC Server Name parameters. If you have several networks on your computer, you must specify the node before the server name, such as Node:Server_Name.

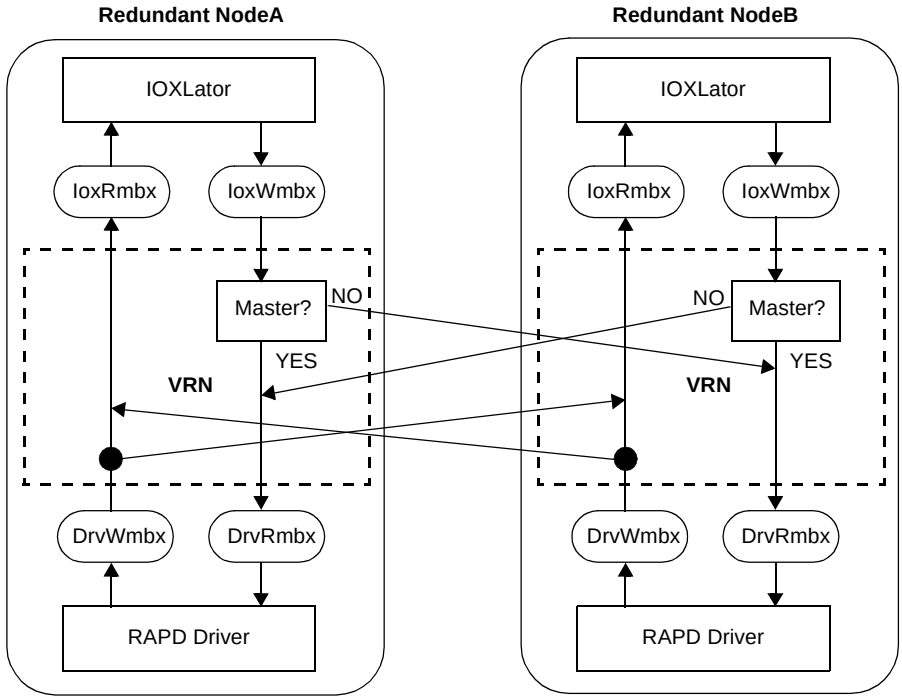
The diagram below shows NodeA as the master running the driver while NodeB's driver is passive. The **VRN_CONTROL** and **VRN_STATUS** tags may be used to enable or disable the read and write tables or the ODX Link Control. The status tag is set to OFF only at startup or if the system is running as a slave. For any other case, the tag is set to ON to start the driver and connect it through the Local link (Mux=1). The master's Status Tag can also be used as an update trigger for a driver that reads unsolicited mailbox data. At normal operation, datasets are sent to the master's ECI through its Local link and to the slave's ECI through the Partner link. If the master fails, the slave takes over and activates its local link and driver.



Note that Mailboxes for ECI (EciRmbx/EciWmbx) and driver (DrvRmbx/DrvWmbx) must be different while IOX cannot be used because of IMX queries. For ECI-based RAPD or ODX, use "Rd/Wr Ds Idx" and then duplicate and rename the ECI Control table entries to be referenced by the driver. For non-ECI based RAPD drivers, make sure the two applications are identical (same dataset tag index) using FLSAVE > FLRESTORE.

REDUNDANT IOXLATOR—IOX/RAPD DRIVER CONFIGURATION

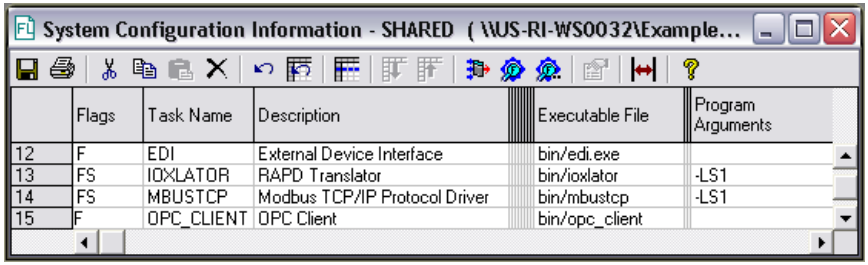
This section describes how best to configure IOXlator and its drivers as a redundant system. The goal is that only the driver on the master communicates with the PLC, while the driver passes its data back to IOXlator both on the master and slave systems. Normally, only the IOXlator running on the server sends reads and writes to the driver. The diagram below illustrates the system.



The following sections show the configuration of a redundant FactoryLink application using mailbox redundancy. Assume that the redundant application resides on nodes Redundant NodeA and Redundant NodeB.

Configure Unique Logical Stations within Redundant System Pair

The first step is to assign a unique logical station to both systems of the redundant pair. This is done by passing a -LS<#> command line argument to both IOXlator and the drivers. The logical station ID should be the same for all tasks on system A, but different from the logical station ID on system B of the redundant pair.



	Flags	Task Name	Description	Executable File	Program Arguments
12	F	EDI	External Device Interface	bin/edi.exe	
13	FS	IOXLATOR	RAPD Translator	bin/ioxlator	-LS1
14	FS	MBUSTCP	Modbus TCP/IP Protocol Driver	bin/mbustcp	-LS1
15	F	OPC_CLIENT	OPC Client	bin/opc_client	

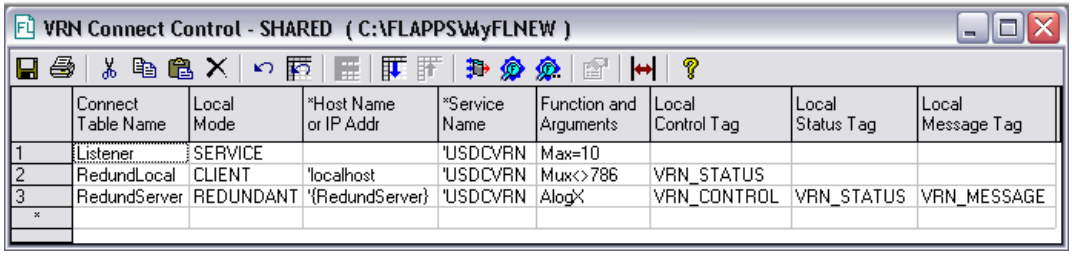
The above example shows both IOXlator and the Modbus RAPD Ethernet driver set to logical station ID 1. On its redundant pair system, both tasks should be set to a logical station ID other than 1, such as 2.

Configure VRN for Redundant RAPD Mailboxes

Next, configure VRN to send IOXlator mailbox messages to the local driver when running on the master and to send mailbox messages to the remote driver running on the slave.

VRN Configuration on Redundant NodeA

The VRN Connect Control table is configured as follows:



	Connect Table Name	Local Mode	*Host Name or IP Addr	*Service Name	Function and Arguments	Local Control Tag	Local Status Tag	Local Message Tag
1	Listener	SERVICE		'USDCVRN	Max=10			
2	RedundLocal	CLIENT	'localhost	'USDCVRN	Mux<>786	VRN_STATUS		
3	RedundServer	REDUNDANT	'{RedundServer}	'USDCVRN	AlogX	VRN_CONTROL	VRN_STATUS	VRN_MESSAGE
*								

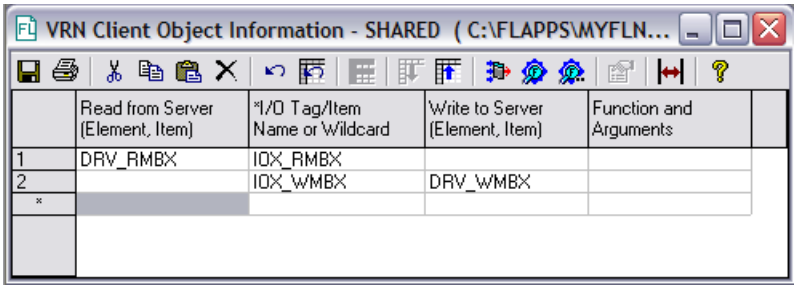
The first entry, table Listener, is required to receive incoming connections.

The second entry, RedundLocal, establishes a client connection to itself, ‘localhost. This connection is only active when the system is not the SLAVE. Local Control Tag has a value of 786 when the local system is a slave, hence the “Mux<>786” function to disable the

connection when the REDUNDANT connection is in SLAVE mode. When the system is the slave, no messages can be exchanged between the local instances of IOXlator and the drivers.

The third entry, table RedundServer, establishes a redundant connection to the node set by the in the {RedundServer} environment variable.

The VRN Client Object Information table is configured as follows for the CLIENT and REDUNDANT modes:



This configuration injects VRN in between IOXlator and its driver on the REDUNDANT communication link or on the local, loopback communication link. Any mailbox messages written by IOXlator are read by VRN and then, if the communication link is active, written back out to the driver receive mailbox.

VRN Configuration on Redundant NodeB

Since the applications are identical on both nodes, the VRN Connect Control table for NodeB is configured exactly the same as the table for NodeA. The only difference is that each system's host file contains the partner's node name or an environment variable is set with the partner's computer name. The FLVRNSetup application object in the Examples Application or the FLNEW template creates the {RedundServer} Environment Variable.

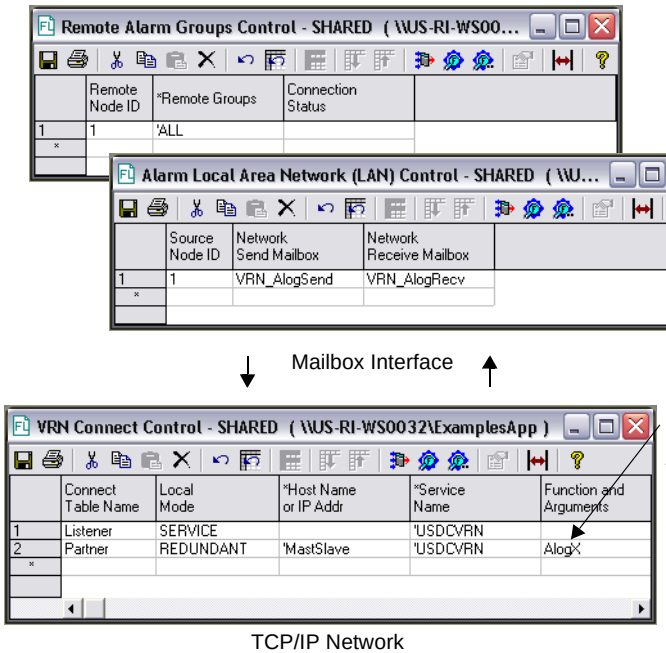
Configure IOXlator to Monitor VRN

Configuring the VRN Local Control field allows IOXlator to monitor the system's master/slave status and ensure that it is initialized with the driver executing on the master. Configuring the read, write, and unsolicited receive disable fields with the VRN Local Control tag VRN_CONTROL disables all control operations from the slave system. This ensures that these operations are invoked solely from the master.

REDUNDANT FACTORYLINK ALARM LOGGER

Below is a sample configuration for a redundant system including the Distributed Alarm Logger. The only entries necessary are the Send/Receive Mailboxes in the Alarm Local Area Network (LAN) Control table, a Remote Node ID pointing to the Source Node in the Remote Alarm Groups Control table, and the ***AlogX*** Function in the VRN Connect Control table. Save the application, then restore it on another computer and run it as a redundant system. The first Redundant system started will be set up as the alarm server (source), while the second will be set up as the alarm client (remote):

Remote Groups, LAN Control, and VRN Connect Configuration at NodeA and NodeB



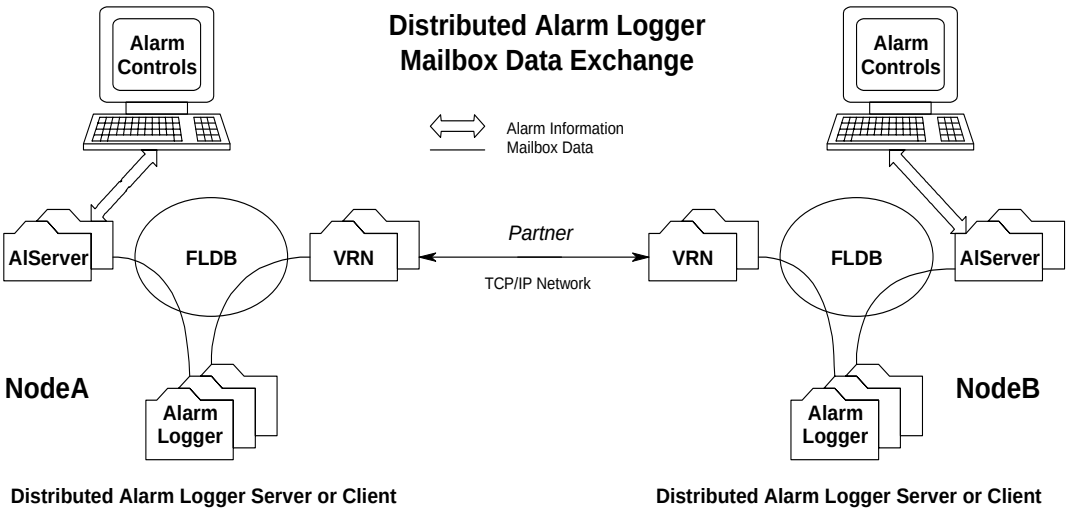
Standard entries for both Distributed Alarm Logger Client and Server

All settings for the Distributed Alarm Logger are standard while the Alarm Definition Information is identical in either system. Note that Unique Alarm IDs are not required, Remote Node ID=255 and IDs 800,000...999,999 are reserved.

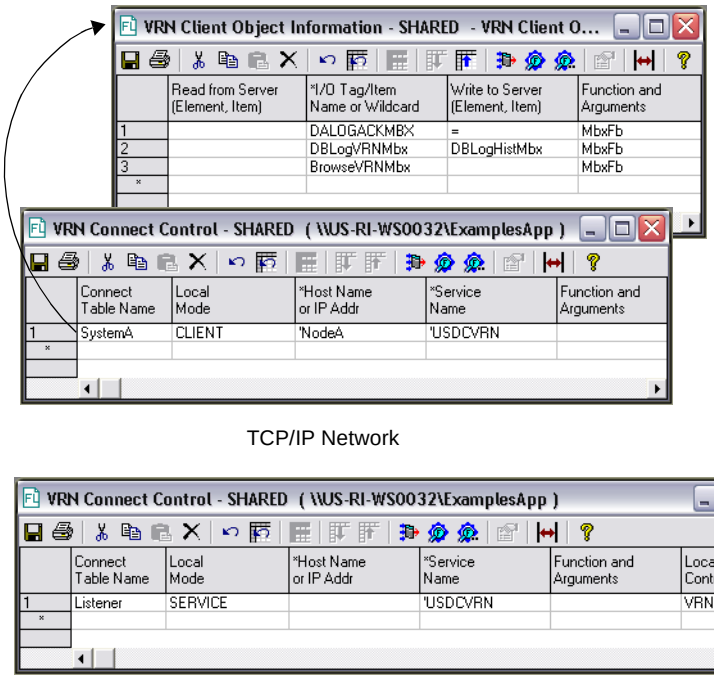
Function *AlogX* (recommended) provides the setup to control the Alarm Logger in either system. Suffix **X** ignores the Alarm Definition Information of the client, that is, the server only uses these tables (see Function Arguments column). All other functions or the system layout may be as on the previous pages.

TCP/IP Network

In the System Configuration table, remove the Alarm Logger's Run Flag since its run status will be controlled by VRN. Also, set the -w program argument to warmstart the AL_LOG task in the event of a restart.


Distributed Alarm Logger
Mailbox Data Exchange
Alarm Information
Mailbox Data
Alarm Controls
Alarm Controls
AlServer
FLDB
VRN
Partner
TCP/IP Network
VRN
FLDB
AlServer
NodeA
Alarm Logger
Alarm Logger
NodeB
Distributed Alarm Logger Server or Client
Distributed Alarm Logger Server or Client

CONFIGURING FACTORYLINK TASKS WITH FEEDBACK MAILBOXES



VRN Client Application

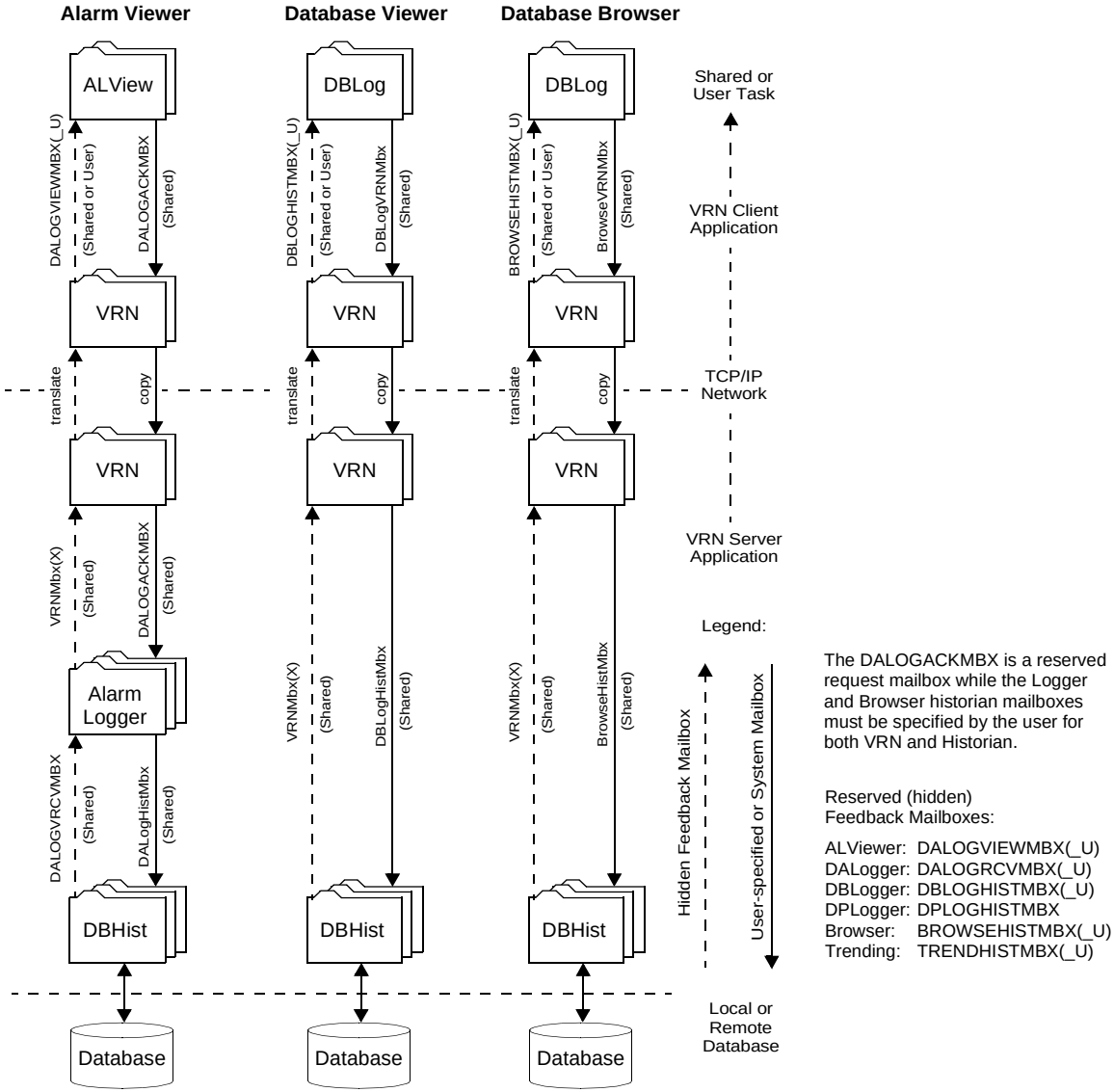
This sample shows three tasks using Feedback Mailboxes: the Alarm Viewer, Database Logger, and Database Browser.

Note that the Alarm Logger in the client must not run when linking Alarm Viewers directly to the server.

VRN Server Application

Note that the Local Control Tag of the *SERVICE* can be used as a Feedback Mailbox buffer. The dimension of the mailbox array VRNMbx[0] must be large enough to accommodate the maximum number of users that can be connected at one time.

The following diagram shows a detailed data flow of all mailboxes involved. The systems may be set up for Shared and/or User tasks. The Alarm historian is shown for information only. Data exchange between Alarm Logger and Historian is standard.



The DALOGACKMBX is a reserved request mailbox while the Logger and Browser historian mailboxes must be specified by the user for both VRN and Historian.

Reserved (hidden) Feedback Mailboxes:

ALViewer: DALOGVIEWMBX(_U)
DALogger: DALOGRCVMBX(_U)
DBLogger: DBLOGHISTMBX(_U)
DPLogger: DPLOGHISTMBX
Browser: BROWSEHISTMBX(_U)
Trending: TRENDHISTMBX(_U)

PROGRAM ARGUMENTS

The parameters listed below can be entered directly with a leading dash in the Program Arguments column of the System Configuration table. Argument names are not case sensitive. You may also reference a file to enter the desired arguments there. In this case, the filename must be specified without a dash in the Program Arguments column. You can apply environment variables using brackets {...} and/or pathnames as required; for example, {flapp}\VRN_para.run.

Argument	Description
-L=<path\logfile>	Log job information.
-V<#>	Set the verbose level for logging. (# = 1 to 4)
-C	Force VRN_init to create all data at startup.
-DefaultMsgLength=<#>	If VRN is the first task that writes to a message tag without a configured length, it sets the max length to # characters. (default=80)
-TagMatchCompatibility	Causes the Wildcard tag matching to use the previous (flawed) algorithm. Tests show that the flaw in the algorithm causes tags to be shared that do not match the wildcard string. The switch is provided to assure that existing applications continue to work, even though they may be sharing more tags than expected. If you use this flag in conjunction with the verbose flag, you will get notified of tags that are being incorrectly accepted by the wildcard comparison algorithm. Example using **-TagMatch Compatibility -V2**: Output: ** Note ** Due to -TagMatchCompatibility switch, '_1_1_1_1' is being incorrectly accepted by the wildcard comparison for '_*_'. Using these together will likely cause a slowdown in performance, but it is worth it to find the places where there were tags being incorrectly included in the VRN transfer. It is recommended that you manually run VRN_INIT as follows: **Vrn_init -C -V2 -TagMatchCompatibility** Study the output and decide whether the tags listed are needed. If they are, you can change the wildcard to include them and then run the application without the -TagMatch Compatibility switch.

Argument	Description
Arguments for VRN Tuning and Performance	
`-ThreadPriority =<#>`	**Caution:** Some of these arguments, if not adjusted correctly, might cause unpredictable results. You may set the VRN Thread Priority and Task Priority Class as described in more detail in the sample file VRN_para.run. However, this should only be adjusted by experts who understand the possible impact to the system. (# =0 to 3, 0 being normal and 3 being the highest)
`-SleepTime=<#>`	Adjust process speed/CPU load by suspending program every scan, in tenth of seconds. (default = 100 ms.)
`-Alive=<#>`	Adjust global alive check time-out, minimum = twice the SleepTime. (default = 60 [s])
`-FirstConnect=<#>`	Time allowed in seconds for first connection when starting in Redundant Mode. (default = 30 [s])
`-ConnDelay=<#>`	Multiple simultaneous connections can be staggered at the rate given by ConnDelay. (default = 3 [s])
`-Throttle=<#>`	Throttle data transmission generally if the internal transmission buffer of 64 kB is full. (# default = 300 [ms])
`-SlaveSyncDelay=<#>`	Data synchronization on the slave of a REDUNDANT system can be delayed to prevent possible overwriting of synchronized data due to a still active but stopped driver at Master_Slave changeover. (default = 3 [s]:)
`-AlogClientDelay=<#>`	Distributed Alarm Logger start delay for Function Alog[..] in a redundant system, this may be useful to unburden the system at REDUNDANT changeover. (default = 5 [s] for client and default = 1 [s] for server)

VRN runs the VRN_init.exe program at startup to prepare all new or changed configuration data prior to running. If required, you can start VRN_init with arguments –aFLAPP –pFLINK -c -v# where FLAPP and FLINK denote the application and program directories, –c is used to recreate all data by requesting a complete initialization at startup, and -v# specifies the verbose level #=1..4.

A sample Program Argument file is shown below:

```
# VRN_para.run Program Argument File
# This is a sample file showing valid program arguments for VRN. If you want VRN to read this
# file, insert the file's name together with its path to the Program Argument column of the System
# Configuration table. Lines beginning with a number sign (#), an asterisk (*), or a space ( ) are
# considered comments; arguments must begin with a dash (–) and may not have spaces.
# You can enable the desired arguments by removing the characters indicating a comment line.
# Arguments for General Purpose and Debugging
# Note that logging may reduce performance due to display or disk access and, it may fill the hard
# disk.
# Log job information, for example, to {FLAPP}\SHARED\LOG\VRN.LOG, default = none:
# –L={FLAPP}\SHARED\LOG\VRN.LOG
# Verbose level -V$ ($=0..4) for logging, the greater the number, the more information is
# displayed, default=1 (medium sensitive):
# –V4
# Force VRN_init to create all data at startup (complete initialization), default = none (changed
# data only):
# –C
# If VRN is the first task that writes to a message tag without a configured length, it sets the max
# length to 80 char (default):
# –DefaultMsgLength=80
# Arguments for Tuning and Performance
# Caution:  Some of these arguments, if not adjusted correctly, might cause unpredictable results.
# You may set the VRN Thread Priority and Task Priority Class as described in more detail in the
# sample file VRN_para.run.However, this should only be adjusted by experts who understand the
# possible impact to the system.
# The Thread Priority 3=Highest, 2=High, 1=Above normal, 0=Normal, may be adjusted to
# withstand CPU load problems, default=0:
# –ThreadPriority=0
# Adjust process speed/CPU load by suspending program every scan, default=100 [ms]:
# –SleepTime=100
# Adjust global alive check timeout, minimum = twice the SleepTime, default=60 [s] (sends a
# signal every 30 sec):
# –Alive=60
# Time allowed for first connect when starting in Redundant Mode, default=30 [s] (may take longer
 if
# no network socket response):
# –FirstConnect=30
# Multiple simultaneous connects can be staggered at the rate given by ConnDelay, default=3 [s]
# connects every 3 sec max:
# –ConnDelay=3
# Throttle data transmission generally if the internal transmission buffer of 64 kB is full,
# default=300 [ms].
# A value of 300 limits the network load to 64kB/300ms or approx. 2 Mbaud:
# –Throttle=300
# Data synchronization on the Slave of a REDUNDANT system can be delayed to prevent
 possible
# overwriting of synchronized data due to a still active but stopped driver at Master→Slave
# changeover, default=3 [s]:
```

–SlaveSyncDelay=3
Distributed Alarm Logger start delay for Function Alog[..] in a redundant system, this may be
useful to unburden the system at REDUNDANT changeover, default=5 [s] for client and default=1 [s]
for server:
–AlogClientDelay=5
–AlogServerDelay=1
Environment Variables and System Tags set at VRN Startup
Similar to the system's environment variables (for example, set by Start→Settings→Control
Panel→System→Environment),
VRN can set multiple tags at startup. In contrast to Math&Logic constants or tag default values.
these so called "System Variables" are not saved and restored with the application but specified
at system setup and can be used in Wildcards and Host Names to identify different systems
running identical applications. The –SetTag argument can be used to set any shared tag(s)
to a Constant or Environment Variable (include braces) as follows:
-**SetTag** (*TagName* =*Constant*)
-**SetTag** (*TagName* ={*EnvironmentVariable*})

INFORMATION AND ERROR MESSAGES

The messages below may be displayed during startup, run-time or shutdown. Additional messages may be logged to file %flapp%\shared\log\VRN.log (%flapp% denotes the FactoryLink application directory). Also, see Program Arguments. The numbers preceding the actual message text are displayed in hexadecimal and transferred to the appropriate Status Tags entered in the VRN Connect Control table. The format specifiers **%s** represent run-time values, **%s:%s** indicates Host and Service names, and **%d** displays an error code returned by the FactoryLink kernel (defined by PAK file FLdefs.h).

Label in VRN.TXT	Task Messages displayed in Run-Time Manager
TE_TERM	**VRN normal shutdown**
TM_RUN	**VRN running**
TM_START	**VRN starting**
TM_DEMO	**#0x0040 No Authorization: Shutdown by timeout**
TE_NOAUT	**#0x0050 Restart prevented: Shutdown FactoryLink first**
TE_NOCT	**#0x0060 Connect CT-File not found (run VRN_init -c)**
NO_AUTHORIZATION	**#0x0070 Option not installed or License not enabled** Check that the VRN Option is installed and the license is enabled. Use the License Wizard to see the purchased options.
TE_ACCESS	**#0x0080 Access: %s Code=%d** (Code=0 configuration error, Code=1..x database access error) A database tag is not configured or an invalid access occurred → please clearly notify the message
TE_VERSION	**#0x0090 Version conflict (VRN_INIT:%s VRN:%s)** → must be same version, re-install VRN

Label in VRN.TXT	Text displayed for Service Message TagArray[x]	Status Tag[x]	
		Analog	Bit1/Bit0
CM_DM_ACT	**#0x0011 SERVICE:%s is active with Service %s**	**17**	**OFF/ON**
CE_DM_SERV	**#0x0042 SERVICE:%s Service:%s not found**	**66**	**ON/OFF**
CM_EX_CLI	**#0x0015 CONNECT%i: External Client %s:%s is active**	**21**	**OFF/ON**
CM_EX_INAC	**#0x0006 CONNECT%i: External Connect %s is inactive**	**6**	**ON/OFF**

Label in VRN.TXT	Text displayed for all other Message Tags	Status Tag	
		Analog	Bit1/Bit0
CM_SR_INAC	**#0x0102 PUBL-CLNT:%s is inactive to %s:%s**	**258**	**ON/OFF**
CM_SR_ACT	**#0x0111 PUBL-CLNT:%s is active to %s:%s**	**273**	**OFF/ON**
CM_SR_RTRY	**#0x0122 PUBL-CLNT:%s tries to connect %s:%s**	**290**	**ON/OFF**
CM_SR_INI	**#0x0132 PUBL-CLNT:%s initializing to %s:%s**	**306**	**ON/OFF**
CE_SR_LIST	**#0x0162 PUBL-CLNT:%s Configuration:%s not found** (run Vrn_init -c)	**354**	**ON/OFF**
CM_CL_INAC	**#0x0202 CLIENT:%s is inactive to %s:%s**	**514**	**ON/OFF**
CM_CL_ACT	**#0x0211 CLIENT:%s is active to %s:%s**	**529**	**OFF/ON**
CM_CL_RTRY	**#0x0222 CLIENT:%s tries to connect %s:%s**	**546**	**ON/OFF**
CM_CL_INI	**#0x0232 CLIENT:%s initializing to %s:%s**	**562**	**ON/OFF**
CE_CL_LIST	**#0x0262 CLIENT:%s Configuration:%s not found** (run Vrn_init -c)	**610**	**ON/OFF**
CM_TA_INAC	**#0x0301 REDUNDANT:%s is inactive to %s:%s**	**769**	**OFF/ON**
CM_TA_SRV	**#0x0311 REDUNDANT:%s is Master (Server) for %s:%s**	**785**	**OFF/ON**
CM_TA_CLI	**#0x0312 REDUNDANT:%s is Slave (Client) of %s:%s**	**786**	**ON/OFF**
CM_TA_FIST	**#0x0320 REDUNDANT:%s first try to connect %s:%s**	**800**	**ON/OFF**
CM_TA_RTRY	**#0x0321 REDUNDANT:%s tries to reconnect %s:%s**	**801**	**OFF/ON**
CM_TA_INI	**#0x0330 REDUNDANT:%s Slave initializing to %s:%s**	**816**	**ON/OFF**
CE_TA_LIST	**#0x0361 REDUNDANT:%s Configuration:%s not found** (run Vrn_init -c)	**865**	**OFF/ON**

Legend:

#0x0000 Inactive for all codes
| | | |
| | | **Ctrl Flags:** Bit0=Enable/Disable if digital Status Tag, Bit1=inverse value
| | **Run Mode:** 0=Off, 1=Running, 2=Connecting, 3=Initializing, 4..7=Error
| **Local Mode:** 0=*SERVICE*, 1=*PUBL-CLNT*, 2=*CLIENT*, 3=*REDUNDANT*
Hexadecimal specifier for message number

The Ctrl Flag Bit0 of the message number is applied when specifying a digital status tag. This can be used to control a task by linking its TASKSTART_S[x] tag to the Status Tag (see “Client, Publ-Clnt, and Redundant State Event Diagram” on page 547). Note that data synchronization at a REDUNDANT slave can be delayed by Program Argument – SlaveSyncDelay to allow for a proper shutdown of drivers and thus prevent possible overwriting of synchronized data.

For a REDUNDANT, CLIENT or PUBL-CLNT connection, the Status Tag can further be used to force the system to a dedicated state, and it can be used as an update trigger for a driver in a redundant system that reads unsolicited mailbox data to master and slave.

Chapter 25. . . .

Waveform Generator and Sequencer

The Waveform Generator and Sequencer (FLWAVE) task provides features for simulating real-world data for the purpose of testing, training, and commissioning of FactoryLink applications and operator stations. The task is divided into three functional areas:

- continuous waveform generation
- event-driven output curve
- event sequencing

The Waveform tables provide the ability to output various continuous waveforms. The waveforms can be used to test or simulate minimum and maximum conditions managed by the HMI/SCADA system.

The Action tables provide an input event-driven output curve. This curve can be delayed to mimic real-world propagation of the data to the IO devices and the corresponding output value changes.

The Sequencer tables provide time or event-driven sequences of digital events. The tables can be chained together to provide for 100’s of steps driving digital tags.

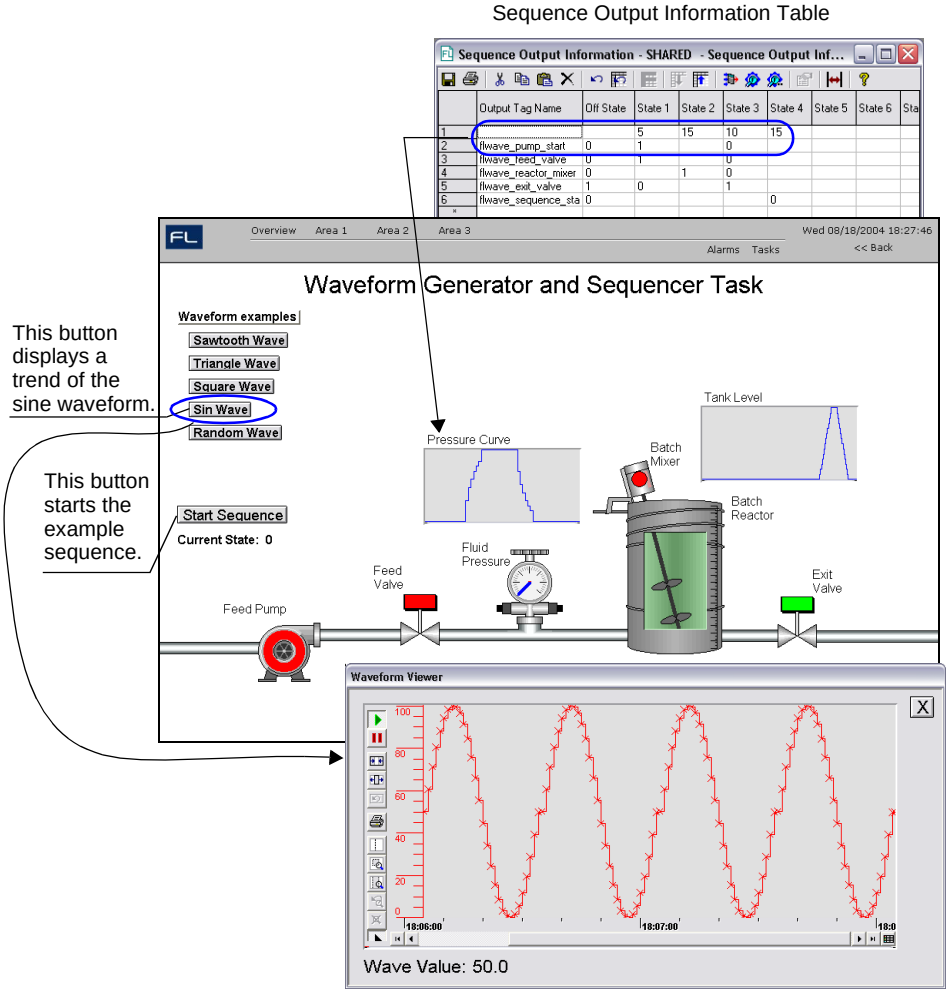
OPERATING PRINCIPLES

This task uses function generators to simulate factory floor data. The function generators include ramp (saw tooth), triangle, sine, square, and random signals that are scaled over a user-defined range and duration. The functions simulate continuous output devices.

Configuration tables are used to establish the simulation. The Waveform tables are used to assign a waveform to a tag, which helps to test boundary conditions in the application and animations. A trigger, which provides the common interface with PLC drivers, is used to activate the waveform.

In Client Builder, an operator can see a snapshot of the waveform as if it were generated from a real device connected to a PLC. Activating the trigger (for example, clicking the Start Sequence button as shown in Figure 25-1) starts the simulation. A sample waveform mimic is available in the Examples Application.

Figure 25-1 Waveform Generator and Sequencer Task Example



The waveform generator is driven by the sequence. The sequence tells the Action Control to begin. Looking at the sample waveform mimic and the configuration tables, Figure 25-1 shows a trigger named flwave_pump_start initiates the pump to start in 5 seconds (State 1 in the Sequence Output Information table), causing the curve to go upwards. After 15 seconds, the pump turns off (State 3), causing the curve to go downwards. The Tank Level is the ramp.

WAVEFORM TABLES

Waveform Control Table

The Waveform Control table specifies the general setup of parameters for the continuous waveform table. The FLWAVE task creates waveforms based on a clock signal. The trigger mechanism acts like a driver Read trigger to sample the waveform at that moment in time.

Accessing

In your server application, open **Other Tasks > Waveform Generator > Waveform Control**.

Field Descriptions

Table Name The name assigned to the table that stores the waveform. The table name is used for the parent/child relationship in the information tables.

Valid Entry: up to 16 alphanumeric characters

Trigger Tag Name The name of the trigger that forces a sample of the curve and transfers the value to the tag specified in the child table. The trigger simulates or matches the triggering for a driver. The values in the child table tag field change continuously, and the application polls the values getting back a value somewhere along the curve. The table is triggered when a non-zero change state occurs.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float

***Table Disable** A digital tag that stops the waveform operation even if the trigger is active. The table is disabled when a value of 1 is in this field or when the value of a tag name is set to ON.

Valid Entry: tag name or constant
Valid Data Type: digital

Clock Tag Name (Optional) The name of a digital tag that provides the rate that the curve is generated. This tag tells the system to generate the point in the background. Each time the trigger tag is set to ON, the waveform steps to the next value internally. The clock tag is triggered when a non-zero change state occurs.

A clock rate value is required when the clock tag is specified. If no value is specified for both the clock tag and clock rate, the time defaults to 1 second.

Valid Entry: tag name
Valid Data Type: digital

Clock Rate (seconds) — (Required when Clock Tag is specified) This value sets the rate (in seconds) that the clock tag will change. Faster clock rates yield a finer resolution in the waveform calculation, but they add processing overhead to the task.

Description — A brief description about the contents or functionality of the waveform specified in the Table Name field.

Valid Entry: up to 64 characters

Waveform Information Table

The Waveform Information table specifies the parameters for a waveform generation. The value of the output tag varies from minimum to maximum in the time specified by the cycle time value. The function generator field determines the shape of the output.

Accessing

In your server application, open **Other Tasks > Waveform Generator > Waveform Control > "your tag name" > Waveform Information**.

Field Descriptions

Output Tag Name — The name for the tag where the current value of the waveform is written when the table is triggered. This tag holds any data tag type (no mailbox tag type).

Valid Entry: up to 16 alphanumeric characters

Function Generator — This key field allows the built-in functions to be used for this tag.

SAW Saw Tooth – Increases the tag value from the minimum value to the maximum value and then drops back to the minimum value to start over. (valid for non-digital tags only)

ISAW Inverse Saw Tooth – Decreases the tag value from the maximum value to the minimum value and jumps back to the maximum value to start over. (valid for non-digital tags only)

TRI Triangle – Increases the tag value from the minimum value to the maximum value and back to the minimum value. (valid for non-digital tags only)

SQR Square – Toggles the tag value from the minimum value to the maximum value each time the trigger is executed.

Digital tags do not need minimum/maximum tags to toggle.

SIN Sine – Approximates the tag value to a smooth sine wave cycling between the minimum and maximum values. (valid for non-digital tags only)

RND Random – Provides a way to introduce "white noise" into a tag value. For digital tags, it will randomly toggle (i.e., not every time). For non-digital tags, the value will be a random value between and including the minimum and maximum values.

Function Constant (Currently used for square and random functions only)

For the square wave, this field specifies the duty cycle. A value of zero is treated the same as a duty cycle of 50%.

For the random function, this value specifies the maximum amount of change between values. If the value is zero, the entire range between the minimum and maximum is used.

Cycle Time (seconds) The interval in seconds for one cycle of the waveform. Do not set this value less than 2 seconds.

For the random function, cycle time specifies the minimum amount of change between values.

***Minimum Value** The minimum value for the waveform.

Valid Entry: tag name or constant

***Maximum Value** The maximum value for the waveform.

Valid Entry: tag name or constant

Noise % The deviation (in percent) between the minimum and maximum values specified. The deviation gets randomly applied to each output. The internal value/shape of the waveform is not affected.

Interlock Tag A tag that disables the output for an action. When the interlock tag value equals the value in the interlock value field, the action is disabled. The output is not affected while the interlock is TRUE. The output value remains at the current value until the interlock tag value no longer equals the interlock value.

Valid Data Type: digital, analog, longana, float

Interlock Value A value used with the interlock tag to designate when to disable the action.

ACTION TABLES

Action Control Table

The Action Control table specifies the control tags for the action information table. When a trigger tag occurs, an action happens.

Accessing

In your server application, open **Other Tasks > Waveform Generator > Action Control**.

Field Descriptions

Table Name The name assigned to the table that stores the action to simulate a waveform and its corresponding outcome. The table name is used for the parent/child relationship in the information tables.

Valid Entry: up to 16 alphanumeric characters

Trigger Tag Name The name of the tag that triggers the capture of the waveform at that point. The trigger simulates or matches the triggering for a driver. The values in the child table tag field change continuously, and the application polls the values getting back a value somewhere along the curve. The table is triggered when a non-zero change state occurs.

Valid Entry: tag name
Valid Data Type: digital, analog, longana, float

***Table Disable** A digital tag that stops the waveform operation even if the trigger is active. The table is disabled when a value of 1 is in this field or when the value of a tag name is set to ON.

Valid Entry: tag name or constant
Valid Data Type: digital

Clock Tag Name (Optional) The name of a digital tag that counts the number of times the waveform is triggered. Each time the trigger tag is set to ON, the waveform steps to the next value internally. The clock tag is triggered when a non-zero change state occurs.

A clock rate value is required when the clock tag is specified. If no value is specified for both the clock tag and clock rate, the time defaults to 1 second.

Valid Entry: tag name
Valid Data Type: digital

Clock Rate (seconds) (Required when Clock Tag is specified) This value sets the rate (in seconds) that the clock tag will change. Faster clock rates yield a finer resolution in the waveform calculation, but they add processing overhead to the task.

Description (Optional) A brief description about the action the waveform is to take.

Valid Entry: up to 64 characters

Action Information Table

The Action Information table specifies the parameters for an action generator. When the input tag is set on, the value of the output tag varies from minimum to maximum in the time specified by the delay time value.

When the input tag is set off, the value of the output tag varies from maximum to minimum in the time specified by the delay time value. The shape of the output is determined by the function generator field.

Accessing

In your server application, open **Other Tasks > Waveform Generator > Action Control > "your tag name" > Action Information**.

Field Descriptions

Output Tag Name The name for the tag where the current value of the waveform is written when the table is triggered. This tag holds any data tag type (no mailbox tag type).

Valid Entry: up to 16 alphanumeric characters

Function Generator The action applied to the output tag. When the input trigger is ON, the output value will transition from the minimum value to the maximum value. When the input trigger is OFF, the value will transition from the maximum to the minimum.

TGL Toggle – After the delay time plus the cycle time, the output will transition from the minimum value to the maximum value.

SAW Ramp – After the delay time, the output will ramp between the minimum value to the maximum value over the time specified by the cycle time.

SIN Sine – The output value follows the 1st 1/4 of a sine wave cycle for the ON event. When the input is OFF, the value follows the 2nd 1/4 of a cycle of the sine wave.

Log Log – The output value follows a logarithmic curve between the maximum and minimum values.

The ITGL, ISAW, ISIN, and ILOG are the inverse actions. For these actions, the output will begin at the maximum value and transition to the minimum value during the ON event, and transition from the minimum to the maximum during the OFF event.

Input Tag Name The initiator of the event. A non-zero value change of the tag causes the action to take place.

Valid Entry: tag name

Delay Time (seconds) The number of seconds to delay before the action begins. This tag simulates the delay of sending a value to a device.

Cycle Time (seconds) The interval in seconds for the output to transition between the minimum and maximum values. Cycle time does not begin counting until the end of the delay time. The value should not be less that two clock ticks, except for the toggle action.

***Minimum Value** The minimum value for the waveform.

Valid Entry: tag name or constant

***Maximum Value** The maximum value for the waveform.

Valid Entry: tag name or constant

Noise % The deviation (in percent) between the minimum and maximum values specified. The deviation gets randomly applied to each output. The internal value/shape of the waveform is not affected.

Interlock Tag A tag that disables the output for an action. When the interlock tag value equals the value in the interlock value field, the action is disabled. The output is not affected while the interlock is TRUE. The output value remains at the current value until the interlock tag value no longer equals the interlock value.

Valid Data Type: digital, analog, longana, float

Interlock Value A value used with the interlock tag to designate when to disable the action.

SEQUENCER TABLES

Sequencer Control Information Table

The Sequencer Control Information table specifies the trigger tags for the sequence.

Accessing

In your server application, open **Other Tasks > Waveform Task Sequencer > Sequencer Control Information**.

Field Descriptions

Sequence Name
The name assigned to the table that stores the sequence order for the waveforms. The table name is used for the parent/child relationship in the sequence information tables.

Valid Entry: up to 16 alphanumeric characters

Sequence Enable Tag Name
The name of the digital tag that starts/stops the waveform operation sequence. When this tag is set to ON, the sequence moves to State 1 and then continues to the next State. When this tag is set to OFF, the sequence moves to Off State.

Valid Entry: tag name
Valid Data Type: digital

Next State Trigger
This name of the digital tag that triggers the waveform value to increment the count for the sequence. Each time the Trigger Tag is set to ON, the sequence continues to the next State until the count is met.

Valid Entry: tag name
Valid Data Type: digital

Current State Tag
This tag displays the current state of the sequence. You can specify a state number to force the sequence to the specified state. Anytime the state changes, the current State is written to this tag. If a state number is specified, the sequence jumps to the specified state, skipping everything in between.

Valid Entry: tag name
Valid Data Type: analog, longana

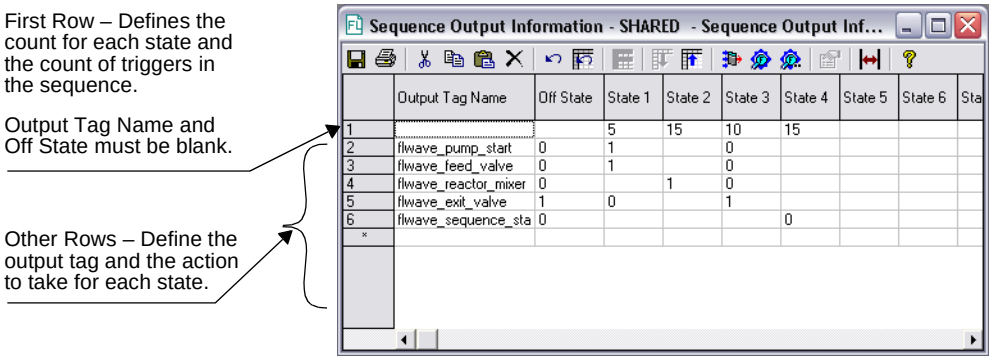
* Table Disable	A digital tag that stops the waveform operation even if the trigger is active. The table is disabled when a value of 1 is in this field or when the value of a tag name is set to ON. Valid Entry: tag name or constant Valid Data Type: digital
Description	(Optional) A brief description about the sequence of the waveform. Valid Entry: up to 64 characters

Sequence Output Information Table

The Sequencer Output Information table contains the output tags and the actions to take for each state of the sequence. The duration of the waveform is the value specified in the first row, and the unit or interval is the value defined by the triggers.

The first row of the table defines the count for each state and the count of triggers in the sequence. For this row, the Output Tag Name and the Off State fields must be blank. Each sequence can have up to 30 steps. The value for each step is the number of the step triggers that will be counted for the step.

All rows after the first row define an output tag and the action to take for each state. A 0 or 1 value specifies what output is written at the beginning of the step. Leaving a field blank indicates no action to take for the state.



Accessing

In your server application, open **Other Tasks > Waveform Task Sequencer > Sequencer Control Information > "your tag name" > Sequence Output Information**.

Field Descriptions

Output Tag Name — The name of the digital tag where the output will be written. The first row of this column must be blank.

Valid Entry: up to 16 alphanumeric characters

Off State — The value that is written for the output tag when the sequence is turned off. When a step is met, the 1 value is written to the output tag.

The first row of this column must be blank. All other rows can have a value.

Valid Entry: **ON** / **OFF**
+ / **–**
1 / **0**
blank – indicates no action to take

State (1 to 30) — The action to take for State *x* of the sequence. In the first row of this column, the value defines the count for the state and the count of triggers in the sequence. In all other rows, the value specifies the output that is written at the beginning of the step.

Valid Entry: **ON** / **OFF**
+ / **–**
1 / **0**
blank – indicates no action to take

ERROR MESSAGES

Error Message	Cause and Action
Corrupt configuration table index **Corrupt data in information panel**	Cause: The flwave.ct file is damaged. Action: Delete this file and then restart the application to rebuild the file.
Error creating trigger list manager **Error inserting into tag list**	Cause: The process could not allocate needed memory. Action: Close any unnecessary windows or programs. If this error message occurs often, depending on the operating system, either add more memory to this system or configure the existing memory differently.
Error reading CT header **Invalid CT header type** **Invalid CT record size**	Cause: Either the flwave.ct file is corrupt or the .CT script file (**/FLINK/CTGEN/flwave.ctg**) and the FactoryLink Run-Time version are not the same version. Action: Delete the .CT file and then restart the application to rebuild the file.
Invalid or missing output tag defined for sequence	Cause: An invalid tag name is specified Action: Delete the .CT file and then restart the application to rebuild the file.
Invalid or missing state change trigger	Cause: Either an invalid value or no value is specified for the sequence Off State. Action: Enter a valid OffState value.
Invalid or missing trigger defined for sequence	Cause: Either an invalid trigger or no trigger is specified for the sequence. Action: Enter a valid tag in the table.
Invalid sequence record data	Cause: The flwave.ct file is damaged. Action: Delete this file and then restart the application to rebuild the file.
Invalid state value	Cause: An invalid value is specified for the sequence state. Action: In the Sequence Output Information table, specify either ON/OFF, +/–, 1/0, or leave blank for the State field.

Error Message	Cause and Action
No tables configured for this task	Cause: The flwave.ct file does not exist, cannot be opened, or is damaged. Action: Delete the .CT file if it exists and then restart the application to rebuild the file.
No triggers are defined for this task	Cause: A required trigger is not defined. Action: Define one or more triggers in the appropriate FLWAVE table.

Appendix . . . .

Format Specifiers

Format specifiers allow you to define the format for all or part of an output string. The following FactoryLink tasks support the use of format specifiers:

- Alarm Supervisor
- Batch Recipe
- File Manager
- Report Generator

Format specifiers permit you to define a variable when a literal is expected. Variable specifiers can consist of two types of objects:

- Ordinary characters, which are copied literally to the output stream
- Format specifiers, which indicate the format in which variable information will display

SYNTAX

% [flags][width][.prec]type

where

%	Always precedes a format specifier.
flags	Controls the format of the output. This can be one of the following. - Left-justified within the field. If you do not specify this flag, the field is right-justified. 0 Fills the spaces to the left of the value with zeros until it reaches the specified width.
width	Specifies minimum field width. For floating point fields, width specifies a minimum total field width that includes the decimal point and the number of digits beyond the decimal point specified with the ".prec" parameter.
.prec	Controls the precision of the numeric field. What precision defines depends on the format type specified by the *type* variable. For exponential (type e) and floating point (type f or g) notations, specify the number of digits to be printed after the decimal point.

For short version of exponential or floating point notations (type g), specify the maximum number of significant digits.

For all other types, specify the minimum number of digits to print. Leading 0s are added to make up the necessary width.

type Specifies the character or numeric type for the value.

d = decimal

s = string

ld = long decimal

e = exponential notation: **[-]m.nnnnnnE[+-]xx**

f = floating-point notation: **[-]mmmm.nnnnnn**

g= use shorter of e or f

u = unsigned decimal

o = unsigned octal

x = unsigned hexadecimal using a - f

X = unsigned hexadecimal using A - F

EXAMPLES

The following table shows examples of valid format specifiers for each FactoryLink data type. For more information about format specifiers, see any ANSI-C reference manual.

Type of Tag	Default Type	Valid Types	Description	Sample Output
Analog	d	d, u, o, x, X	%04d specifies a right-justified decimal value with a minimum field width of four digits. The 0 specifies the value is padded with zeros.	0005 0015 0150 2400
			%3u specifies a right-justified unsigned decimal value with a minimum field width of 3 digits. The value is padded with spaces.	5 15 150 2400
			%-3u specifies the same as the example above, except the hyphen (-) before the width specifies the value is left-justified.	5 15 150 2400

Type of Tag	Default Type	Valid Types	Description	Sample Output
Long Analog	ld	d, ld, u, o, x, X	%-7ld specifies a left-justified long decimal value with a minimum field width of seven digits. The value is padded to the right with spaces.	5 15 2400 1000000
			%05ld specifies a right-justified long decimal value with a minimum field width of five. A zero before the width specifies the value is padded with zeros.	00005 00015 02400 1000000
Floating-point	f	e, f, g	%6.2f specifies a right-justified floating-point value with a minimum total field width of 6 digits. (The decimal point counts as 1 digit.) This means two digits are displayed after the decimal point and at least three digits are displayed before the decimal point. The value is padded with spaces.	5.51 150.08 24000.65
Message	s	s	%5s specifies a right-justified message string with a minimum field width of five characters. The value is padded with spaces.	on off alarm
			%-5s is the same as the example above, except the hyphen (-) before the width specifies the value is left-justified.	on off alarm
Digital			Specifies a character string to display depending on the digital value. A typical use might be to indicate a valve status of ON or OFF rather than a 1 or 0. To do this, enter the desired character strings for each condition (one string for each possible value, 0 and 1) in this format: **open | closed** where open specifies the message open to print when the tag value is 1 (ON) closed specifies the message closed to print when the tag value is 0 (OFF)	on off

- **A | FORMAT SPECIFIERS**
- *Examples*
-
-

Index

D

E

F

G

H

P

T

U

V

W