FactoryLink 6.6.0

. . . .

PowerSQL User Guide

FactoryLink 6.6.0 / September 1998 / 000A*DOC-SQL*000*660

•

•

•

•

Table of Contents . . . .

WebClient User Guide

-
-
-
-

Preface

PURPOSE

FactoryLink 6.6.0 PowerSQL (Structured Query Language) works in conjunction with the FactoryLink Historian task to allow an application to access data in an external relational database through a result window. In addition, PowerSQL processes SQL statements that are entered in a FactoryLink Message tag.

This guide presents the technical information necessary to install, configure, and administer PowerSQL.

AUDIENCE

The primary audience of this manual is application developers, programmers, or administrators who are involved in providing PowerSQL.

STRUCTURE OF THE *POWERSQL USER GUIDE*

The *PowerSQL User Guide* is an Additional Manual in the FactoryLink Documentation Set. Refer to the Preface in *Fundamentals* for the structure of the entire Documentation Set.

This manual consists of three chapters:

- Principles of PowerSQL operation
- Configuring PowerSQL
- PowerSQL status codes and status messages

HOW TO USE THIS MANUAL

The material in this manual is presented sequentially in performance order. We recommend you read the entire manual to familiarize yourself with the complete procedure before you proceed to develop your application.

Valid Entry:	standard tag name
Valid Data Type:	analog

PowerSQL at a Glance

Located at the beginning of this manual is a section named “PowerSQL at a Glance.” This section provides a quick key to locations to find information to perform the procedures detailed in that part with hypertext links to those locations.

CONVENTIONS

The material in the Documentation Set adheres to the guidelines published in *The Digital Technical Documentation Handbook* by Schultz, Darrow, Kavanagh, and Morse; *Developing International User Information* by Jones, Kennelly, Mueller, Sweezy, Thomas, and Velez; and corporate style guidelines.

This manual uses the following conventions.

Convention	Description
...	Horizontal ellipsis points indicate the omission of material from an example. The information is omitted because it is not important to the topic being discussed.
. . .	Vertical ellipsis points indicate the omission of information from an example or command format. The information is omitted because it is not important to the topic being discussed.
italic type	Italic type is used to denote user-supplied variables in command examples. Italic type also sets off references to specific documents.
`monospace type`	Monospace type is used to denote command names and code examples or example output.
`bold monospace type`	Bold monospace type is used in command examples to indicate words that must be typed literally.
sans serif type	Sans Serif type is used to set off field names, button names, and keys on the keyboard.

Convention	Description
press nnnnn	Press is used to denote a key on the keyboard. The key name will appear in a sans serif type.
click on nnnnn	Click on is used to denote a button on the screen. The button name will appear in a sans serif type.
Shift+F1	The + indicates the keys must be pressed simultaneously. Shift+F1 indicates you hold down the Shift key while you press another key or mouse button (indicated here by F1). Other key combinations are presented in the same manner.
F1 F2 F3	The space between the key callouts indicates press and release. The key sequence F1 F2 F3 indicates you press and release F1, then F2, and then F3. Other key combinations are presented in the same manner.
File>Open	The > indicates a progression through a menu sequence. File>Open indicates you choose Open from the File menu to perform the required action. Other menu sequences are presented in the same manner.

Convention	Description
FLAPP\user\drw\mydrw.g	The \ indicates the directory structure for the listed file. FLAPP\user\drw\mydrw.g indicates the drawing file mydrw.g is located in the drw sub-directory of the user sub-directory to the FLAPP directory. Other directory structures are presented in the same manner.
[ ]	Brackets indicate an optional argument. You can choose none, one, or all of the options.
{ } and |	Braces indicate a choice. You must choose one of the elements. The vertical bar separates choices within braces.

Example Syntax

Example syntax using these conventions is provided below:

`command` *input_file* [*input_file*...] {*a* | *b*} *output_file*

where

`command`	is typed as it is displayed in the syntax.
input_file	indicates a variable the user supplies.
[*input_file*...]	indicates the user can optionally supply multiple input file names, each name separated by a space.
{*a* | *b*}	indicates either the a or b must be specified as an argument.
output_file	indicates the user must specify an output file.

GETTING HELP

Contact your Sales or Customer Support Representative for help with troubleshooting problems.

Also, help files are included for each configuration panel. Click on Help on the panel menu bar to access these files.

. . . .

PowerSQL at a Glance

Using PowerSQL User Guide	
For details on performing the following steps...	**Go to...**
1. Read about the general operating principles of PowerSQL User Guide	,*“Principles of PowerSQL Operation*”
2. Read about the use of logical expressions with PowerSQL User Guide	,*“Principles of PowerSQL Operation*”
3. Read about the use of program arguments with PowerSQL User Guide	,*“Principles of PowerSQL Operation*”
4. Configure PowerSQL User Guide Control Panel	“PowerSQL Control Panel”
5. Configure PowerSQL User Guide Information Panel	*“PowerSQL Information Panel”*
6. Reference PowerSQL User Guide status codes and status messages	,“PowerSQL Status Codes and Status Messages”

-
-
-
-

Principles of PowerSQL Operation

The FactoryLink PowerSQL (Structured Query Language) task works in conjunction with the FactoryLink Historian task to allow an application to access data in an external relational database through a result window. In addition, PowerSQL processes SQL statements that are entered in a FactoryLink Message tag. 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 PowerVB or Math & Logic.
- Allows you to execute database-stored procedures for database servers that support them

For information on PowerSQL Commit Logic, please refer to page 17.

PRINCIPLES OF OPERATION

This chapter introduces the operational concepts behind the principles of PowerSQL operations. Refer to Chapter 2, “Configuring PowerSQL,” for information on procedures to configure this task.

In db2, as **opposed** to Oracle, the slightest variation of row uniformity, (two rows are detected as being too similar in makeup,etc.) will result a negative status for all other rows.

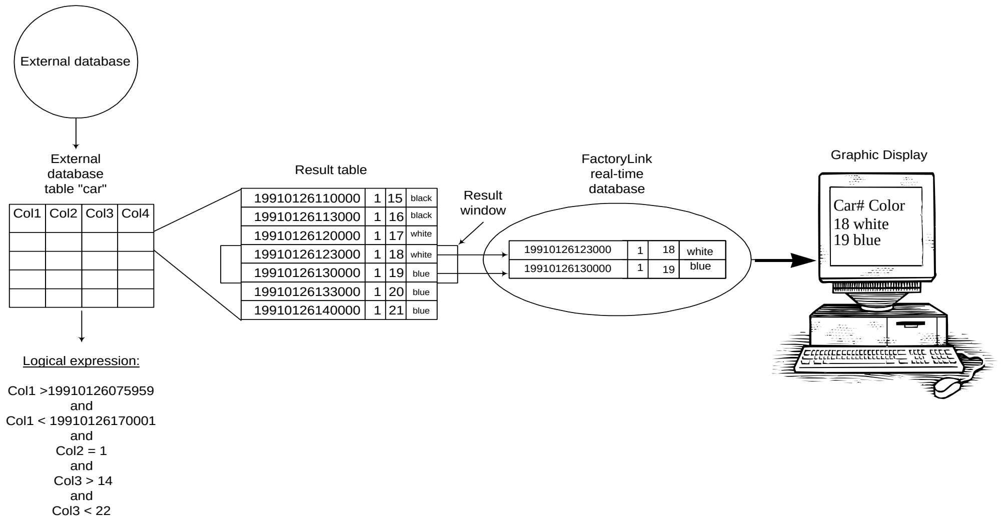
PowerSQL is a Historian-client task that communicates with Historian through mailbox tags to send and receive historical information stored in an external database using SQL.

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.

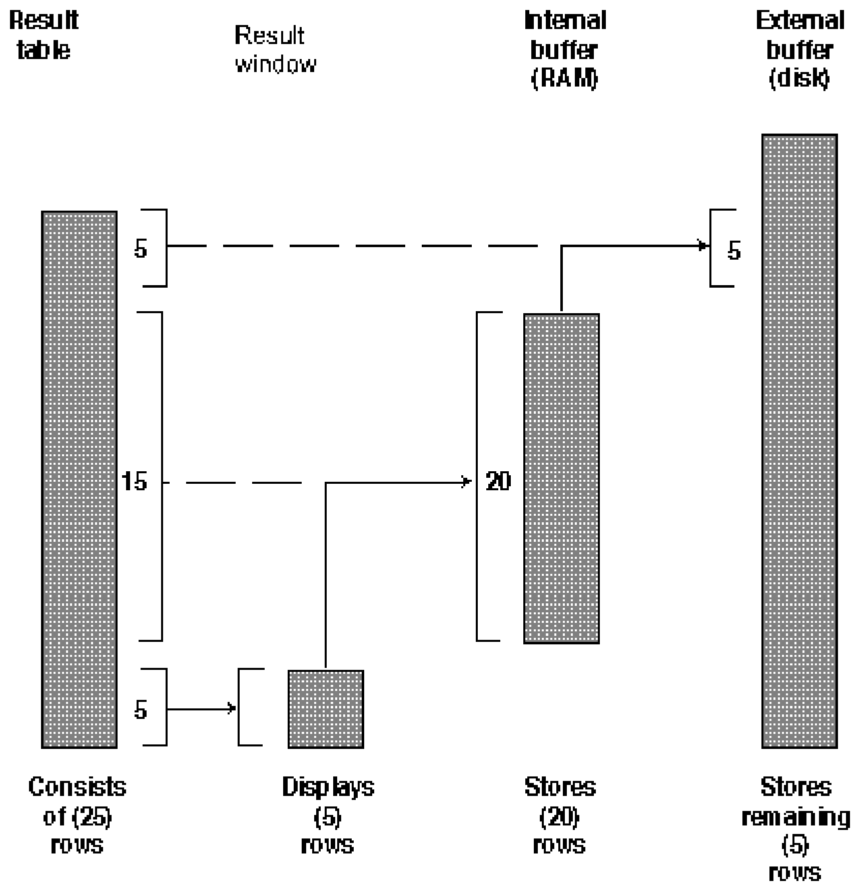
The relationships among the external database, the result table, the result window, the real-time database, and the graphic display are displayed below.


External database
External database table "car"
Col1 Col2 Col3 Col4
Logical expression:
Col1 >19910126075959 and Col1 < 19910126170001 and Col2 = 1 and Col3 > 14 and Col3 < 22
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
19910126140000 1 21 blue
Result window
FactoryLink real-time database
19910126123000 1 18 white
19910126130000 1 19 blue
Graphic Display
Car# Color
18 white
19 blue

PowerSQL 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. The buffers are shown in the following illustration.

Table 1-0



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 PowerSQL, a logical expression is a command containing a standard Structured Query Language (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. See any SQL guide, such as *Quick Reference Guide to SQL* and/or the user manual for the relational database in use for information about writing SQL statements.

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.

Table 0-1

TRANDATE	CONVEYOR	CARNUM	COLOR
19910126080000	1	9	yellow
19910126083000	1	10	red
19910126090000	1	11	red
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

Table 0-1

TRANDATE	CONVEYOR	CARNUM	COLOR
19910126133000	1	20	blue
19910126140000	1	21	blue
19910126143000	1	22	brown
19910126150000	1	23	burgundy
19910126153000	1	24	blue
19910126160000	1	25	blue
19910126163000	1	26	brown
19910126170000	1	27	burgundy

A sample WHERE clause referencing the previous table CAR is

```
TRANDATE > ‘19910126075959’ AND TRANDATE < ‘19910126170001’ AND
CONVEYOR = 1 AND CARNUM > 14 AND CARNUM <22
```

In this example, the WHERE clause requests:

```
What colors cars 15 through 21 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.

Table 0-1

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

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	Via a message tag.

CONFIGURING PROGRAM ARGUMENTS

Configure the following system configuration program arguments to affect PowerSQL functionality:

-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 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[4-160] or -s[4-160]	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.
-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.
-W[5-300] or -w[5-300]	Sets the maximum timeout in seconds for PowerSQL to wait for a response from the Historian task. The default is 30 seconds.
-C[0-2] or -c[0-2]	In the previous version of the PowerSQL tasks, a COMMIT statement was performed after all database accesses, except the SELECT statement, were executed as a non-dynamic SQL statement. This included the execution of stored procedures and dynamic SQL statements through the use of the SQL tag. The execution of dynamic SQL statements, especially for stored procedures, can result in very complex database operations that include many separate steps. In such cases, the PowerSQL task

does not have at its disposal the necessary information to determine if a COMMIT should be executed or whether a ROLLBACK is more appropriate. This has the potential to COMMIT unwanted database updates in the case of execution failures. Proper procedures would dictate that COMMIT/ROLLBACK logic should be programmed into the stored procedures. Therefore, it has been decided that procedurally the PowerSQL task should not execute a COMMIT after the execution of dynamic SQL statements. However, since this would have an impact on an existing application, the task has been modified to accept a new program argument that will control the COMMIT logic. The new argument (entered in the Program Arguments field of the System Configuration table for the PowerSQL task) is:

```
-cN
```

where N is a modifier to select various levels of COMMIT logic. The default action (no argument listed or "-c2") will be COMMIT logic exactly as in the previous version, and thus no modifications are required to existing applications. The argument "-c1" will result in no COMMITs for dynamic SQL statements. The non-dynamic SQL operations (traditional insert, delete, and update statements) are followed by a COMMIT. The argument "-c0" will result in no COMMITs for any statements executed, except for a final COMMIT upon task shutdown.

Even though the default configuration is to continue to perform a COMMIT after dynamic SQL statements, USDATA strongly recommends that application be modified to use the "-c1" argument and that all stored procedures be updated to include all necessary and appropriate COMMIT/ROLLBACK logic. Of course, if all stored procedures currently contain such logic, then the execution of a COMMIT by the PowerSQL task has no effect, and negates the need to include the "-c1" argument.

Use of the "-c0" configuration is not particularly recommended, since failure to COMMIT non-dynamic 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 does offer users the ability to take full responsibility for COMMIT logic away from the PowerSQL task and have it become part of the application design and control.

Chapter 2 . . . .

Configuring PowerSQL

This section provides a field-by-field description of the configuration panels used to configure the PowerSQL task. PowerSQL uses one configuration table: the PowerSQL configuration table, which consists of the following two panels:

- PowerSQL Control panel—Refer to “PowerSQL Control Panel” on page 20 for more information.
- PowerSQL Information panel—Refer to “PowerSQL Information Panel” on page 36 for more information.

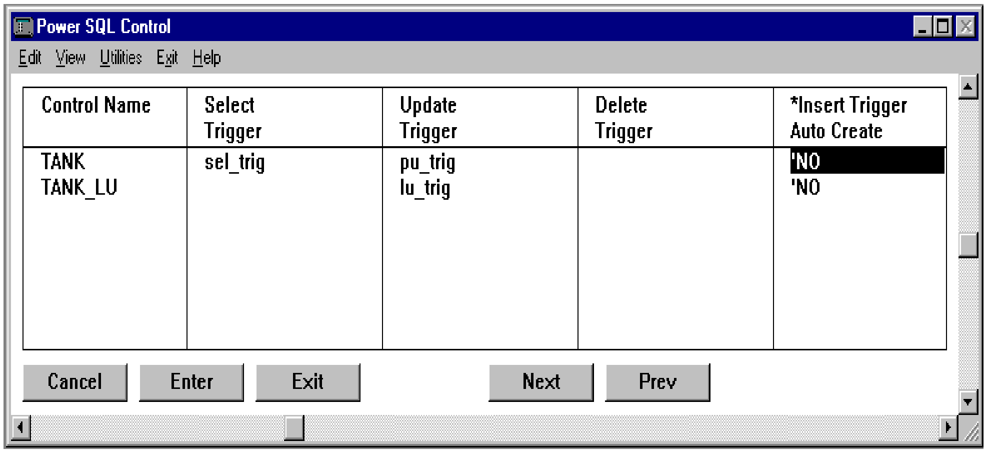
These panels are configured in the SHARED or USER domains.

PowerSQL Control Panel

Perform the following steps to configure the PowerSQL Control panel:

1 Ensure the appropriate domain is selected in the Configuration Manager Domain Selection box. PowerSQL is most commonly configured as a USER domain task but it can also be configured as a SHARED task.

2 Choose PowerSQL in the Configuration Manager Main Menu to display the PowerSQL Control panel.



Control Name	Select Trigger	Update Trigger	Delete Trigger	*Insert Trigger Auto Create
TANK	sel_trig	pu_trig		'NO
TANK_LU		lu_trig		'NO

3 Specify the following information for this panel:

- For a control record using Insert Trigger, do not use Select, Update, or Delete Triggers.
- For a control record using Power SQL tag, use only either Select or Update Trigger.
- For a control record not using Insert Trigger or Power SQL tag, use the Select Trigger alone or with Update Trigger or with Delete Trigger or with both Update and Delete Triggers.
- For a control record not using Insert Trigger or Power SQL tag or Select Trigger, use Update Trigger and/or Delete Trigger.

Control Name	Alphanumeric string of 1 to 15 characters that specifies the developer-assigned name of the control record. Valid Entry: alphanumeric string of 1 to 15 characters

Select Trigger	Name of a 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 panel and places it in a result table for you to view or manipulate.

If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type.

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

Update Trigger

Name of a 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. Refer to FactoryLink Configuration Guide Chapter 18, "Database Logger," for more information on configuring the Database Schema Creation panel. Refer to "Working with Tables" on page 273 of the FactoryLink Configuration Guide if you need to create a unique index on a dBASE IV table that already exists. Consult the appropriate RDBMS user's manual if you need to create a unique index on a non-dBASEIV database table that already exists.

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 panel. 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 panel 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 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-duplicate index key. However, further testing with various ODBC drivers behave differently to this situation as noted in the following list:

MS Access97 ODBC Driver- inserts rows only until the error is encountered and aborts the rest of the operation. Status information is only valid up to the row that failed.

MS SQL Server 6.5 Driver- inserted data is correct. Duplicate row is not inserted. However, the driver does not return an error message that allows for proper update of the status for that row.

Intersolv Sybase Driver- same as MS SQL Server 6.5 Driver results.

Intersolv Oracle Driver- works as desired.

Intersolv Informix Driver- works as desired.

IBM DB/2 version 2.1.2 and ver 5.0 Driver- once an error is encountered, it aborts the entire set of operations. The failure status is properly returned for all rows.

If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type.

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

Delete Trigger	Name of a 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.

For a positional delete to work, the database table must have a unique index. This can be configured in Database Schema Creation or when the database table is created. Refer to Chapter 11, "Defining Schemas," of the FactoryLink Configuration Guide for more information on configuring the Database Schema Creation panel. Refer to "Indexing" on page 80 in Chapter 5 of the FactoryLink ECS Fundamentals Guide if you need to create a unique index on a dBASE IV table that already exists. Consult the appropriate RDBMS user's manual if you need to create a unique index on a non-dBASE IV database table that already exists.

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 panel. 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 panel 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.

If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type.

Valid Entry:	standard 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. Configuring a select trigger with an insert trigger is an invalid configuration that is reported at task startup. The insert trigger causes PowerSQL to construct an insert SQL statement based on the information entered in the PowerSQL Information panel. PowerSQL can insert one row or multiple rows of data when the insert SQL statement is executed. 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 panel 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. If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type. Valid Entry: YES or NO or standard tag name Valid Data Type: digital, analog, longana, float, message
Move Trigger	(Requires use of the Select Trigger.) Name of an analog 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.

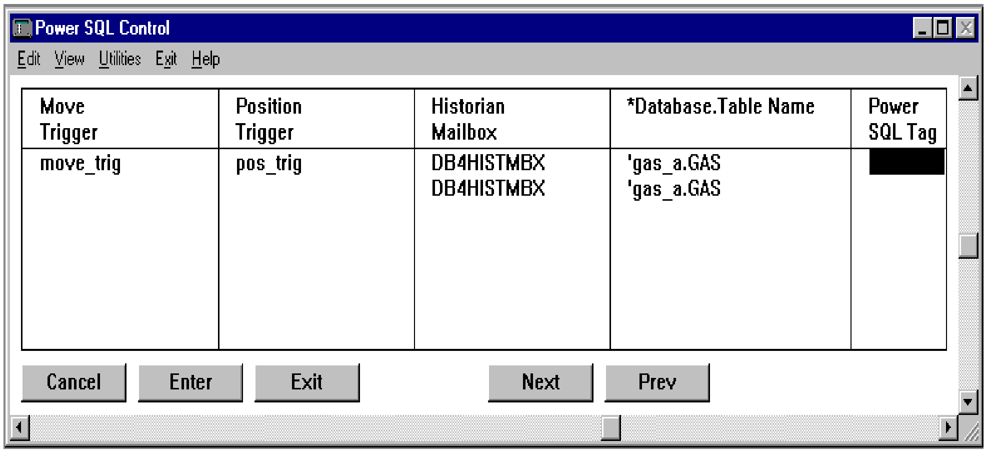
For example, if the value of the Move Trigger tag is 3 and the Current Row tag is 0 (active row is the first row in the result window) and the result window size (data array size) is five rows, the current row tag is changed to 3 and the data in the result window is not scrolled. If the Move Trigger 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. This is so 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.

Move operations can be performed only on result tables; therefore, move operations cannot be performed unless you have defined and executed a select trigger.

If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type.

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



Move Trigger	Position Trigger	Historian Mailbox	*Database.Table Name	Power SQL Tag
move_trig	pos_trig	DB4HISTMBX	'gas_a.GAS	
		DB4HISTMBX	'gas_a.GAS	

Position Trigger	(Requires use of the Select Trigger field.) Name of a 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.

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. This is so 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.

Position operations are performed only on result tables; therefore, position operations cannot be performed unless you define and execute a select trigger.

If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type.

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

Historian Mailbox	Name of a 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.

If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type.

Valid Entry:	standard tag name
Valid Data Type:	mailbox

*Database.Table Name	This field can be configured as a real-time database tag or as a string constant. 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. This is so 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.

The user can choose to enter a string constant. If so, the string constant must have 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.

If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type.

Valid Entry:	alphanumeric string of 1 to 63 characters or a standard tag name
Valid Data Type:	message

PowerSQL Tag

Name of a message tag that is used by the user 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 database server that the Historian task is communicating with. 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 panel. The SQL statement can also generate a result table and each result data column must also have an associated record in the PowerSQL Information panel. See the description of the Column Expression field in the PowerSQL Information panel for more detail. For SQL statements that do not require an input variable or generate a result table, the PowerSQL Information panel can be left empty.

Note: You may 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,) the user should not include a ";" at the end of the SQL statement.

If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type.

Valid Entry:	standard tag name
Valid Data Type:	message

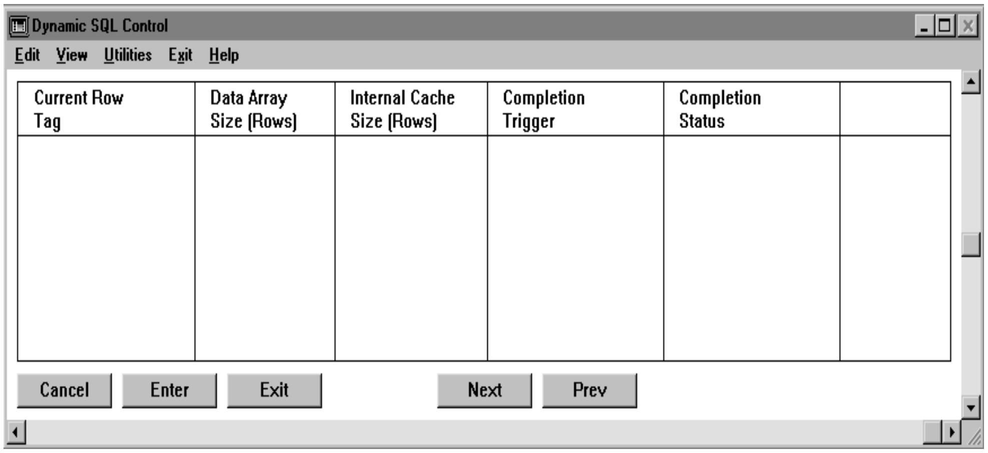
Current Row Tag — Name of a real-time database tag. For select, move, or position operations, this tag's value indicates the position of the active row of data in a result window. After PowerSQL performs a Select, Move, or Position operation, PowerSQL writes the value indicated by the position of the 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.

PowerSQL performs all positional update and positional delete operations on the row indicated by the current row tag if a select trigger is defined.

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 panel must be contiguous, since PowerSQL reads the tag specified in the information panel and the next current row tags in the tag array when a Logical Update, Logical Delete, or Insert operation is executed.

If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type.

Valid Entry:	standard tag name
Valid Data Type:	analog



Current Row Tag	Data Array Size (Rows)	Internal Cache Size (Rows)	Completion Trigger	Completion Status	

Data Array Size (Rows)	Number between 1 and 1000 that 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 panel 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: number between 1 and 1000

Internal Cache Size (Rows)	Number between 0 and 9999 that specifies the number of rows of data in a result table that are cached. This field makes sense only for select operations that have a move and/or a position trigger configured. So 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. Refer to Move Trigger and Position Trigger field descriptions for more information. 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, setting the data array size to more than 50 is not recommended. Viewing more than 50 rows of information in a table grid is difficult. 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, setting data array size to a large value > 50 makes sense. 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: numeric value of up to 9999
Completion Trigger	Name of a tag whose change-status flag is set when any operation undertaken by this control record is completed. If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type. Valid Entry: standard tag name Valid Data Type: digital, analog, longana, float, message

Completion Status	Name of a tag whose value 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. Refer to "Click on Exit to return to the Main Menu." on page 43 for the codes and messages that can display in this tag.

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 in , "PowerSQL Status Codes and Status Messages." 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 must be capable of storing values determined by the Data Array Size field.

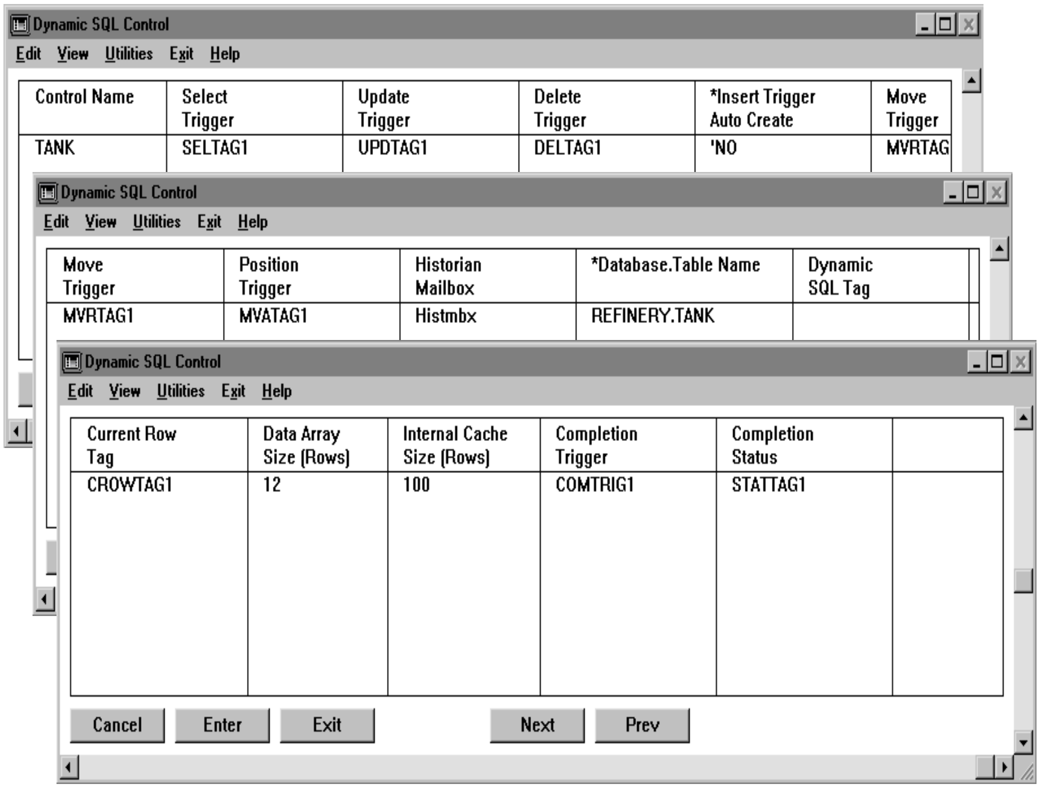
You can configure this tag to work in conjunction with output objects in the Application Editor task to display codes or messages on any graphics screen. Refer to Chapter 10, "Animating Input/Output Fields" in the FactoryLink ECS Application Editor Guide for information about defining output objects.

You can also configure Math & Logic to monitor this tag and respond to or ignore errors that occur.

If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type.

Valid Entry:	standard tag name
Valid Data Type:	analog, longana, message

The panel resembles the sample panel shown in the following illustration when you have specified all information.



Control Name	Select Trigger	Update Trigger	Delete Trigger	*Insert Trigger Auto Create	Move Trigger
TANK	SELTAG1	UPDTAG1	DELTAG1	'NO	MVRTAG

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 Status	
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 element CROWTAG1 when PowerSQL performs a select, move, or position operation. The Completion Status element STATTAG1 contains status information about the operation just completed. The change-status flag for the digital element COMTRG1 is set when an operation for this result window is complete.

Because the Auto Create Record? 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.

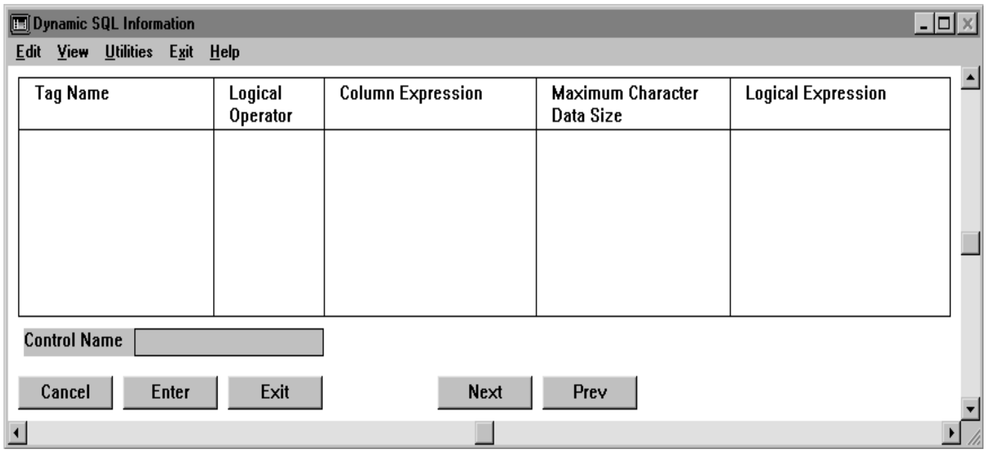
Because the Data Array Size is 12, the result window can display 12 rows of data from the result table at a time. The internal cache can hold 100 rows of data from the result table.

4 Click on Enter to save the information.

5 Click on Next to get to the PowerSQL Information panel.

POWERSQL INFORMATION PANEL

Use the PowerSQL Information panel to configure the details of the SQL operation defined in the PowerSQL Control panel. Specify the following information to configure the PowerSQL Information panel:



Tag Name	Name of a 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 panel 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. If the tag specified in this field is not already defined, a Tag Definition dialog is displayed when you click on Enter. Select the appropriate data type.

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

Logical Operator

The Logical Operator field is ignored for control records that have a PowerSQL tag configured. 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. 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	Alphanumeric string of 1 to 63 characters that specifies: (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. Include table, if the table name is different from the table name specified in the *Database.Table Name field in the PowerSQL Control panel. column — is the name of the column within the relational database table. You can use the same column name in two rows of a panel. 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 FactoryLink 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: alphanumeric string of 1 and 63 characters
Maximum Character Data Size	Limits 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: numeric value between 1 and 256
Logical Expression	The Logical Expression field is ignored for control records that have a PowerSQL Tag configured. The Logical Expression field is 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.

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 Manager 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. 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 using the dBASE IV Historian, limit the logical expressions to this list of relational operators. If not using the dBASE IV Historian, consult the RDBMS SQL Language user’s manual for more information.

The WHERE clause is generated by appending the Logical Operator, Column Expression, and Logical Expression fields in the order displayed in the PowerSQL Information panel. 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 panel, 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 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: Options 1 and 3 are different. The result is the same for both options, but option 3 allows the user to change the tagSQLExpression 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

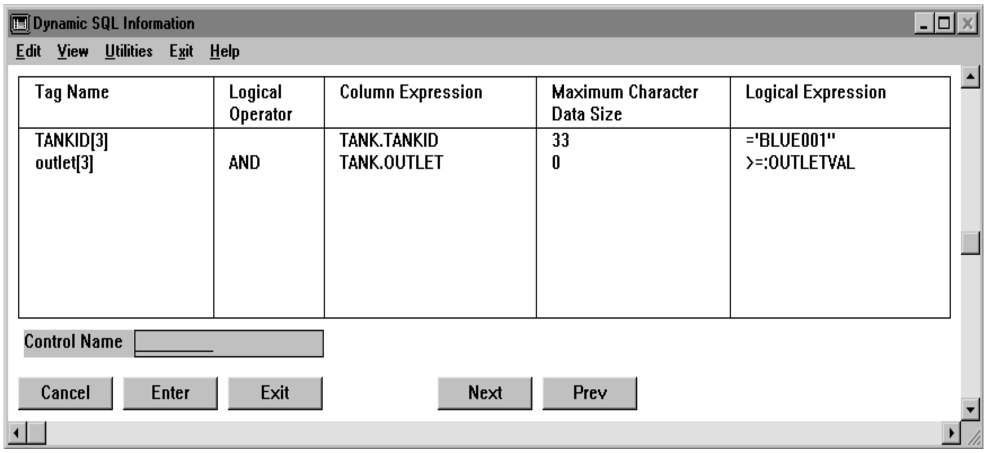
generates the following WHERE clause:

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.

The panel resembles the following sample panel when complete.



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 panel) 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 a Tag Name tag array has been defined for TANKID and OUTLET, the values in the columns the logical expression is true for are displayed in the result window.

6 Click on Enter to save the information.

7 Click on Exit to return to the Main Menu.

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 PKGCSP03;

/

Create or replace package body PKGCSP03 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 PKGCSP03;

/
commit;
```

Note: USDATA is not responsible for any changes in Oracle. Please refer to the Oracle manual for any changes.

Chapter 3 . . . .

PowerSQL Status Codes and Status Messages

When an error occurs at run time in PowerSQL, the Historian, or another Historian-client task, FactoryLink sends a status code or status message for display to the Run-Time Manager screen and to the Completion Status tag. FactoryLink also sends a longer, more descriptive message to the log file if you created a log file. This section describes the error messages that can be displayed on the Run-Time Manager screen for the PowerSQL task.

The codes and messages are displayed on a graphics screen if you define an output text object to display them.

The following table lists status codes written to a numeric Completion Status tag defined in the PowerSQL Control panel. The descriptive message is written to a message Completion Status tag and may also be written to the Task Message tag in the System Configuration panel. See "Historian Messages" on page 258 of the FactoryLink ECS Reference Guide for status codes smaller than 100. The Historian generates these codes, and they are returned to PowerSQL 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 panel 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 tag names have been defined to hold the data from the select operation.	Define some tag names in the Tag Name field in the PowerSQL Information panel.
103	No fields for insert.	The task is trying to execute an insert operation, but no tag names have been defined to hold the data from the insert operation.	Define some tag names in the Tag Name field in the PowerSQL Information panel.
104	No fields for update.	Task is trying to execute an update operation, but no tag names have been defined to hold the data from the update operation.	Define some tag names in the Tag Name field in the PowerSQL Information panel.
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. A select operation must be executed before an update operation.	Execute a select operation and then retry the update operation.
107	Cannot delete until select is performed.	A select trigger is defined, but a select operation has not been executed. A select operation must be executed 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. A select operation must be executed 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 nonexistent row	No action required.
110	A FactoryLink function returned an error.	FactoryLink PAK function encountered an unknown or unexpected error.	Contact Customer Support.
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. If there is enough disk space, contact Customer Support.
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 PowerSQL.
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 PowerSQL.
115	Can’t open log file.	Disk space too low.	If disk space is low, delete unneeded files or programs; otherwise, contact Customer Support.

116	A request for memory failed.	Internal error.	Contact Customer Support.
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.
118	PowerSQL Information record has an invalid configuration.	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.	Change the information record.
119	Tag array is too small for PowerSQL operation.	A tag referenced in the information panel is not large enough to contain Data Array Size values.	Change the dimension for the tag or enter another tag large enough to contain Data Array Size values.

One of the following messages is displayed to the right of SQLTASK on the Run-Time Manager screen if an error occurs with PowerSQL or Historian at run time. The first three letters in the message are a variable that indicates whether the message came from PowerSQL or from the Historian (HIS). This variable three-letter prefix displays in the messages below as nnn. Open the .LOG file to display the complete message if it is truncated on the Run-Time Manager screen.

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 panel. Enter the correct name in the Historian Mailbox field of the PowerSQL Control panel.

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 in a FactoryLink task other than PowerSQL if you did not make a typographical error.

nnn*-[CT_HDR] No sel, upd, or delete trigger defined. Control name: *name

Cause:	None of the following triggers is defined: - select - delete - update
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 panel.
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 FactoryLink PAK function encountered an unknown or unexpected error.
Action:	Contact Customer Support.

nnn-[FL_FUNC] Function 'FL_WRITE' returned error 9. Control name: *control 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:	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 panel.

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 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 data from more than ten unique databases.
Action:	Reference fewer than ten 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.

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.

Cause: An error occurred within PowerSQL. Contact Customer Support.

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 Customer Support.

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 panel.

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 panel.

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 panel.

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 non-existent 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 panel and/or specify a logical expression.

nnn*-[NO_UNIQ_INDX] Can’t 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] Can’t open .LOG file.**

Cause:	The computer may have run out of disk space.
Action:	Delete any unnecessary files or programs. Contact Customer Support.

nnn*-[SEL_B4_DEL] Can’t 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 then retry the delete operation.

nnn*-[SEL_B4_MOVE] Can’t 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] Can't 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 then 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 database administrator for the external 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 panel to create a correct SQL statement if the error is a syntax error. Ensure the database table exists if the panels 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 in the Configuration Manager Main Menu. 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. Contact Customer Support if this is not the problem.

F_BAD_RMBX Invalid global mailbox: *mailbox name*

Cause: PowerSQL's predefined mailbox tag is nonexistent in the GLOBAL.CT file.

Action: Contact Customer Support.

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 is 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 Can't open CT archive *file name*

Cause: A .CT file may have been deleted.

Action: 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.

***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 before result table start.
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 direction because all rows deleted in that direction.
Action:	No action required.

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 operation.

nnn-[DYNAMIC_COLUMNS] Only $OUTPUT, $INPUT, $INOUT column expressions allowed. Control name: _name_

Cause: A Column Expression field in information panel 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 that is contained in a PowerSQL Tag.

Action: Only 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 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 and 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 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 the Data Array Size values can be stored in the tag array.

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 that 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 and 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 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 and ensure that the tag is defined in the application.

nnn-[NO_OBJ_CT] Cannot access OBJECT CT

Cause: Internal error; cannot access the Tag Name database.

Action: Contact Customer Support.

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 somewhere else in the application.

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 the dimensions of tag is correct based on the tag definition.

nnn-[UPDCOL_EXPR] Column assignment _column expression_ cannot have other fields configured. Control name: _name_

Cause: For column assignment:
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 Customer Support.

nnn-[FL_FUNC] Error writing tag. Error = _error code_. Control name: _name_

Cause:	The fl_write FactoryLink PAK function call encountered an unknown or unexpected error.
Action:	Contact Customer Support.

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 are is configured in the PowerSQL information panel.
Action:	Add more input records to the PowerSQL information panel 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 is configured in the PowerSQL information panel.
Action:	Add more output records to the PowerSQL information panel or change the SQL statement in the PowerSQL tag.

nnn-[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 panel.
Action:	Contact Customer Support.

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 panel.
Action:	Contact Customer Support.

•

•

•

•

Index