Optimize LCN Data Access

L61570

LCN

TotalPlant

Notices and Trademarks

This module supports **TotalPlant** Solution (TPS) system network.

TPS is the evolution of TDC 3000^{X}.

Honeywell Inc.
Industrial Automation and Control
Automation College
2820 West Kelton Lane
Phoenix, AZ 85053-3028
1-800 852-3211

Table of Contents

Figures and Tables

Acronyms

RNOS ..... Real-time Network Operating System
PSDP ..... Processor Status Data Point
ID ..... Identifier
IDB ..... Intermeditate Database
DDT ..... Data Definition Table

Parameters

ALENBST ..... Alarm Enable Status

Introduction

Module Overview

About this module

Data Access is a subsystem located between RNOS and the application. It is part of the network communication layer of the software environment that is found in all LCN nodes. These three elements of Data Access provide the link between application requestors and data owners and are discussed in this module:

1. Parameter Access,
2. Name Conversion, and
3. Queries.

This module describes the ways the user can more efficiently use Data Access to improve LCN performance.

Objective

Given data access optimization techniques, you should be able to configure schematics, history requests, and data requests from an upper level processor that minimizes the LCN Data Access work load.

Optimize LCN Data Access

Definitions

Function Set

A set of tasks within a physical node that can interact with each other to perform a well defined function (Examples: Control; Area Manager).

Data Realm

Data upon which a set of tasks work (Examples: Unit_01; Area_01).

Logical node

A logical node is defined by a Function Set and a Data Realm. In the majority of cases, a logical node is completely contained within a physical node. The major categories for logical nodes are

- PSDP for node n
- AM or CG for unit n
- US/GUS for area n
- HG for hiway n
- NIM for network n
- $UNITOPS point:
  - contains only the ALENBST parameter—an array of the alarm enable state status of all units,
  - resident in HGs, NIMs, AMs, and CGs.

Example 1

A typical HG has three logical nodes:

- the PSDP,
- the hiway, and
- the $UNITOPS point.

Example 2

Because each AM unit is a logical node, an AM can have many logical nodes:

- the PSDP,
- AM units 1 through n (for an AM controlling unit 02, its logical node is Control.Unit_02),
- the $UNITOPS point.

Data Owner

A Data Owner is a logical node that supplies requested data to other logical nodes through the Data Access request interfaces:

- Parameter Access,

- Name Conversion, and
- Query.

HGs and NIMs do not initiate data requests on the LCN, but only respond to data requests from other nodes. They are referred to as "pure" LCN data owners. Other data owners include AMs and CGs.

Parameter Access

What is Parameter Access?

One of the three elements of Data Access is Parameter Access. Parameter Access is a common transport mechanism for LCN data. It resides in the software environment, providing an interface between the operating system network communication mechanism and the application functions.

Parameter Access allows the application functions to ignore, to the fullest extent possible, the source and form of the data they import and the destination and use of the data they export. The applications call on Data Access, and Data Access does the work of determining the source or destination of the data.

Use of logical nodes for Parameter Access

Parameter Access operates on the basis of logical nodes.

All entities have an internal entity ID containing the data owner's logical node. Using the logical node identifier, Data Access determines the node to get data from or to store data to and divides data request lists into separate messages on a logical node basis.

Intermediate databases (IDBs)

To do its work, Parameter Access creates intermediate databases (IDBs). IDBs are temporary self-defining databases used to locally store values from around the network. IDBs are created in heap of the requesting node and are deleted when no longer needed by the node.

Parameter Access creates IDBs and uses them to

- build lists of entities and associated parameters that are
  - to be gathered from the network, or
  - to be scattered across the network.
- access individual items from the list.

Users cause IDBs (lists) to be created when they

- add data requiring LCN update into US/GUS custom schematics,
- build history groups in the History Module, and
- build Data Definition Tables or point lists in the Computer Gateway.

Collection Sets

A Collection Set is a group of read-only parameters in a single entity that can be accessed by specifying a "macro" parameter name. The application data owner provides the access with special Data Access support.

Collection Sets

- are functionally organized,
- work only in conjunction with IDBs (IDBs provide the structure on which to attach Collection Sets to access individual parameters from the set).

The individual parameters in a Collection Set

- are accessed by the IDB access mechanism.
- depend on the type and structure of the entity.

Collection Set—Group display example

A Collection Set for a Group display contains the parameters needed to support the display for all entities of a particular type and structure.

Certain operating displays are variable in that they depend on the type and structure of the entity selected for display. Variable operating displays include the Group, Process Module, and Detail displays.

The US/GUS uses IDBs with Collection Sets to perform display updates. To the requesting US/GUS, a Group display update request appears as a 1-parameter request, because all of the parameters are included in one Collection Set. The data owner, however, responds with 25 to 30 parameters.

The use of Collection Sets skews the Data Owner node's parameter per second value ($PRSTSnn.PARSEC) on the low side. PARSEC will increment only to indicate the "macro" parameter, not the 25-30 parameters with which the Data Owner responds.

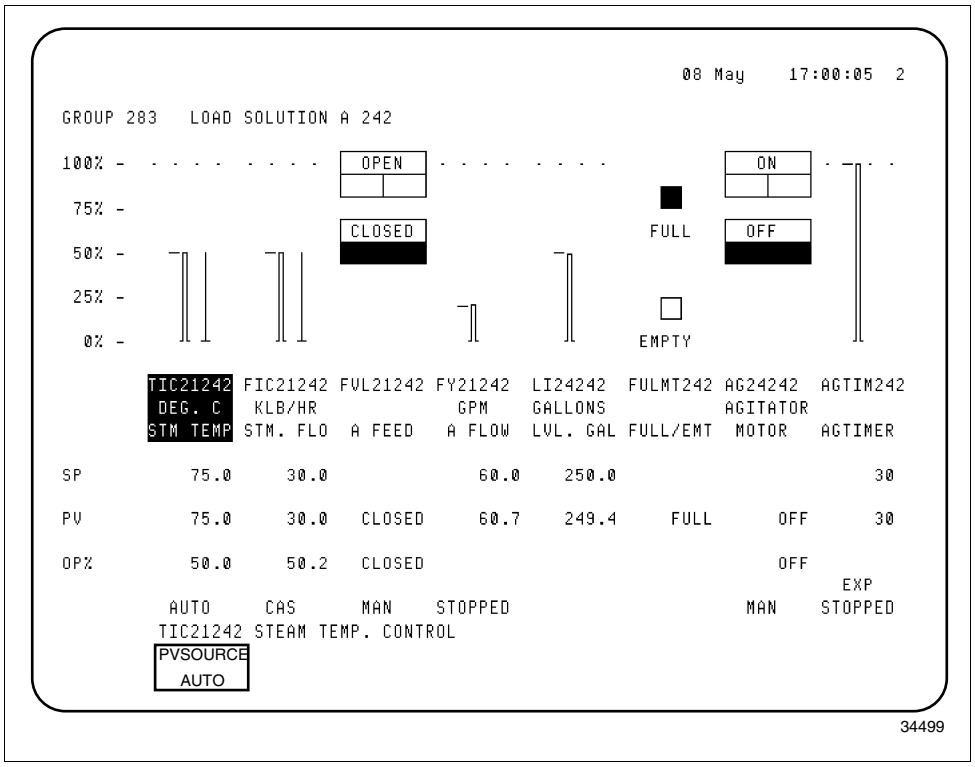


Figure 1 – Group Display—Collection Set Example

Collection Set—Detail display example

The US/GUS calls up a Detail display differently than a Group display. For different entities of the same type, the user can configure different parameters and they must be displayed at the detail display; consequently, the Detail display is broken into separate Collection Sets.

Upon callup of a Detail display, the US/GUS obtains standard information from the data owner by using Collection Sets; however, within the display type used for a particular Detail display, blank fields exist into which the data owner can place nonstandard parameter and data information (if the point has been configured for such)

Obtaining data in this way reduces the number of standard display types required.

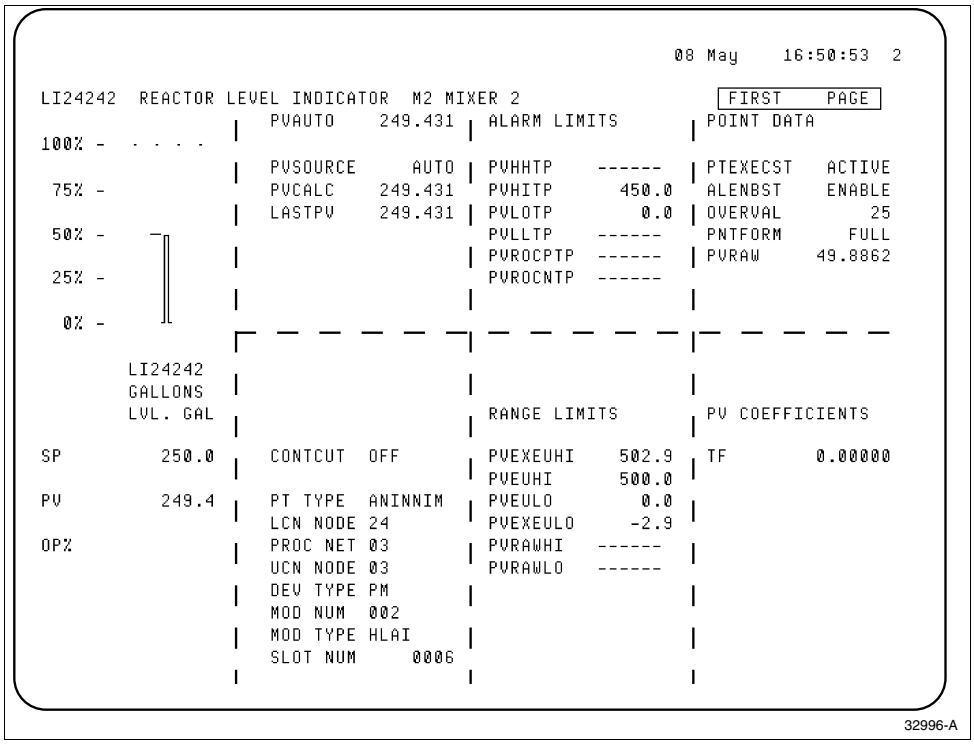


Figure 2 – Detail Display—Collection Set example

IDB examples

Table 1 describes examples of IDBs commonly used on the LCN.

Table 1 – IDB Examples

Node Type	IDB Example
History Module	The HM builds an IDB for each of three history groups (60 point.parameters per IDB).
Universal Station Global User Station	The US/GUS uses IDBs through the use of • collection groups, standard display updates (collection sets) such as group and detail displays, and the Data Entity Builder in the engineering personality.
Application Module	The AM accesses data on a periodic basis, using IDBs for prefetching and poststoring data.
Computer Gateway	The CG accesses LCN data through data definition tables (DDTs) or point lists. These upper-level processor requests require IDBs.

Name Conversion

Name conversion

One of the three elements of Data Access is name conversion. Names exist in two forms:

- external (human readable)—called the name,
- internal—called the ID.

Data Access converts names to IDs and vice versa.

Types of names

Names that are converted by Data Access include

- entity names,
- parameter names,
- enumeration names, and
- parameter list names.

Where are name conversions performed?

Most name conversions are performed locally by the requesting US/GUS, AM, CG, or NG; an exception is entity name conversion:

- Entity name conversion—an exception because Entity names are distributed across the physical nodes on the network. These conversions are performed by the owning node.

Queries

Query operations

Data Access supports several types of query operations. In general, query operations involve the caller specifying an IDB size. Data Access uses IDB size to determine the number of items to return.

Query types

Table 2 describes the three general types of queries.

Table 2 – Types of Data Access Queries

Query Type	Description
History Queries	Data Access accepts a list of entities and parameters in IDB form, distributes them to the appropriate HM service nodes, and returns the history data collected by the HMs. Examples: • PV Retrieval selection from System Menu, • Trends, and • History retrieval requests from the AM or CG.
Point Queries	Data Access returns the entity IDs and a set of parameters for each entity specified by box, physical node, or unit. These types of queries can be generated by certain selections from the System Menu or from the Documentation Tool.
Parameter Queries	Data Access returns the data form of all the parameters on the specified entity. These types of queries are used for Control Language (CL) program compiles and schematic building.

Presorting by Logical Node

Description

To optimize Data Access, the user can set up data requests so that Data Access has to do less work and therefore will gain some performance reward. The performance rewards are related to the efficient use of lists.

Recall that Data Access creates, from lists (IDBs) of point.parameters, separate LCN messages by logical node; therefore, when Data Access is asked to store or read a list, it sorts the list by logical node.

If the sorting task can be reduced, performance will be improved.

If the user knows in what logical nodes the data resides, the lists can be built and presorted by logical node in order to save Data Access sort time.

Example 1

Problem: Assume you want to fetch three parameters p1, p2, and p3 from three AM entities A1, A2, and A3. Entities A1, A2, and A3 are each assigned to separate units 1, 2, and 3.

Solution: Because each point in the list belongs to a different logical node, presorting the list by parameter increases Data Access sorting time and therefore is inefficient. It is more efficient to sort the list by point (logical node).

Sorted By Parameter	Sorted By Logical Node
A1.p1 unit 1	A1.p1 unit 1
A2.p1 unit 2	A1.p2 unit 1
A3.p1 unit 3	A1.p3 unit 1
A1.p2 unit 1	A2.p1 unit 2
A2.p2 unit 2	A2.p2 unit 2
A3.p2 unit 3	A2.p3 unit 2
A1.p3 unit 1	A3.p1 unit 3
A2.p3 unit 2	A3.p2 unit 3
A3.p3 unit 3	A3.p3 unit 3

Example 2

Problem: Entities A1 and A3 are assigned to unit 1; A2 is assigned to unit 2.

Solution: In this case, the most efficient use of the list is to sort by logical node.

Sorted By Parameter	Sorted By Logical Node
A1.p1 unit 1	A1.p1 unit 1
A2.p1 unit 2	A1.p2 unit 1
A3.p1 unit 1	A1.p3 unit 1
A1.p2 unit 1	A3.p1 unit 1
A2.p2 unit 2	A3.p2 unit 1
A3.p2 unit 1	A3.p3 unit 1
A1.p3 unit 1	A2.p1 unit 2
A2.p3 unit 2	A2.p2 unit 2
A3.p3 unit 1	A2.p3 unit 2

Reducing the Number of LCN Messages

Description

Because Data Access creates an LCN message for each logical node in a list, LCN performance gains can be realized by organizing lists to reduce the number of LCN messages. This may reduce the work load on the data owners and decrease LCN traffic.

Sorting example

Say your database is organized into two lists of AM points:

LIST 1	LIST 2
half of the points are from unit 1 half of the points are from unit 2.	half of the points are from unit 1 half of the points are from unit 2

Without Presorting: Collecting the lists as defined results in Data Access creating at least four LCN messages, one message per unit per list:

LIST 1	LIST 2
Unit 1 Unit 2	Unit 1 Unit 2

With Sorting: If possible, it would be better to reorganize the lists so that LIST 1 has only unit 1 points and LIST 2 has only unit 2 points; this reduces the number of LCN messages by half:

LIST 1	LIST 2
Unit 1	Unit 2

Parameters per LCN message

The operating system imposes a limit on the number of words of data that can be moved in a single message (6000 words). To avoid violating this limit, Data Access breaks up an LCN message into 200 parameter pieces. For example, if Data Access is asked to collect a list with 500 parameters from the same logical node, Data Access will build three LCN messages:

Message 1 200 parameters
Message 2 200 parameters
Message 3 100 parameters

If you were to graph Data Access time per parameter as items were added to a list, you would see jumps in the graph each time a 200 parameter boundary was crossed. This is because the additional overhead of another LCN message has been incurred.

Parameter limit example

The first list requires an additional LCN message because the parameter limit was exceeded even though only one logical node is represented.

LIST 1	LIST 2
300 parameters, Unit 1 (two LCN messages)	100 parameters, Unit 2 (one LCN message)

Presorting CG Data Definition Tables

Data Definition Tables

The Computer Gateway requests data from the LCN by use of user-built Data Definition Tables (DDTs). These are lists of point.parameters. Each DDT can be configured to collect up to 300 parameters (except history DDTs, which can have only 24 parameters). Recall that each LCN request can have a maximum of only 200 parameters.

When the CG receives a request using a DDT, it sends the entire DDT to Data Access, which then breaks up the list into requests to be sent over the LCN to the data owner nodes.

Sorting by logical node

The DDTs can be presorted by logical node to reduce the time required for the Data Access sorting task. Thus, by the time Data Access receives the list its work load has been reduced.

Organize to reduce messages

Data Access translates IDBs into LCN messages. The number of LCN messages can be reduced by properly organizing the points in the DDTs.

Examples:

400 AM point.parameters from two units are to be fetched.

1. The following DDT organization results in three LCN messages:
   DDT1: 200 points from unit 1; 100 points from unit 2
   DDT2: 100 points from unit 2

2. The following DDT organization results in two LCN messages:
   DDT1: 200 points from unit 1
   DDT2: 200 points from unit 2

Organizing the points by unit reduces LCN traffic and may improve the receiving node's response time by reducing the number of messages that it has to service.

Setting Up History Groups in the HM

Sorting by logical node

Data Access creates one IDB for each of three history groups (60 point.parameters per IDB). If all 60 point.parameters are from the same logical node, the Data Access sorting task is reduced and only one LCN message results.

Reduce data owner load

It is also desirable to attempt to reduce the load on the data owner. This may be necessary in cases where the HGs or NIMs are trying to accommodate a heavy schematic load in addition to the history requests. In this case, the user must balance the needs of Data Access with the needs of the Data Owners.

For example, one way to reduce data owner load is to set up the history groups so that they contain points from different data owners.

Example

Problem — It is desired to historize point.parameters in the same unit from an AM, NIM, and HG.

Solution — Set up these groups:

- group 1 with AM points,
- group 2 with NIM points,
- group 3 with HG points.

This causes Data Access to create three LCN messages of 20 point.parameters per data owner; but, each data owner is required to service only 20 point.parameters at a time.

If points from all three data owners are combined into one history group, the load reduces on each data owner as well, but the required Data Access sorting time increases.

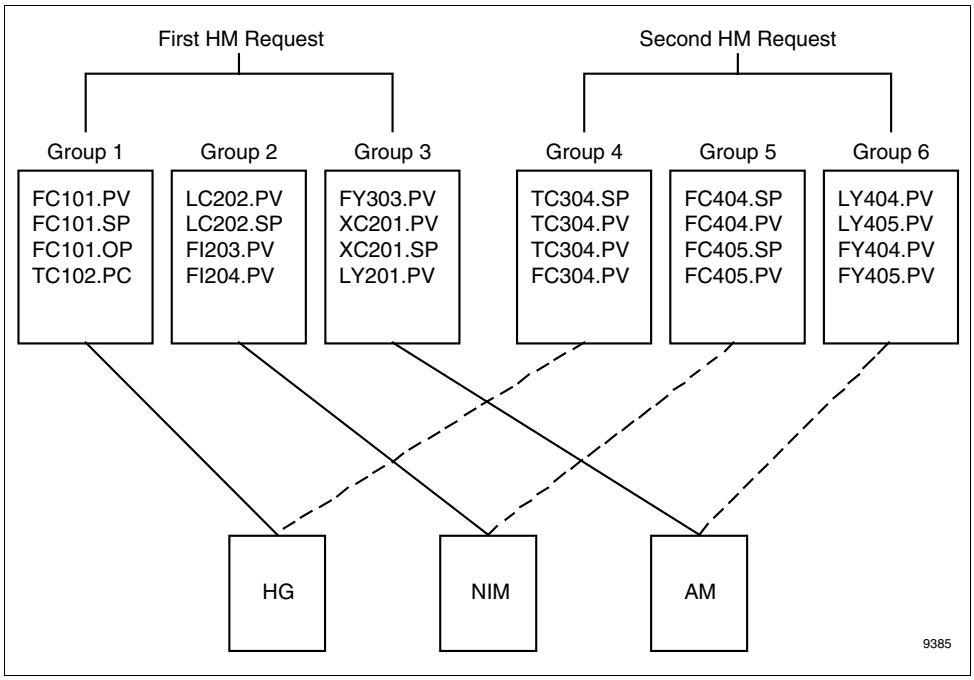


Figure 3 – Setting up History Groups

Optimizing Custom Display Data Access

Description

The impact of a custom display on overall system loading (the data owners and the LCN in general) can be minimized by reducing the rate of data gathering. This can be accomplished through the use of collection groups.

Collection groups are used for sorting Data Access requests. Variables should be sorted into collection groups by

1. Collection rate
2. Logical node
3. UCN control-resident and I/O-resident parameters

Collection group

A collection group is a list of variables used in a custom display that provides a means to sort the custom display variables by collection group ID and collection rate.

Collection groups are defined using the Set Collection command in the Picture Editor or the Data Collect pull-down menu in the GUS display builder. They can be identified by numbers ranging from 0 to 245, FST (for half-second updates using the FAST key) and CZ (for change zone variables). There can be 127 Collection Groups per custom schematic.

Collection rate

Collection rates for each collection group range from 0 to 255 in multiples of 4 seconds as follows:

Rate Number	Actual Collection Rate
0	Update on call up
1	Update every 4 seconds
2	Update every 8 seconds
.	.
.	.
255	Update every 17 minutes

The default collection rate, if collection groups are not specified by the user, is four seconds for all variables.

Sorting by logical node

Each collection group is a list for which an IDB will be created. Presorting the lists (creating collection groups) by logical node reduces the Data Access sort time required to process the request.

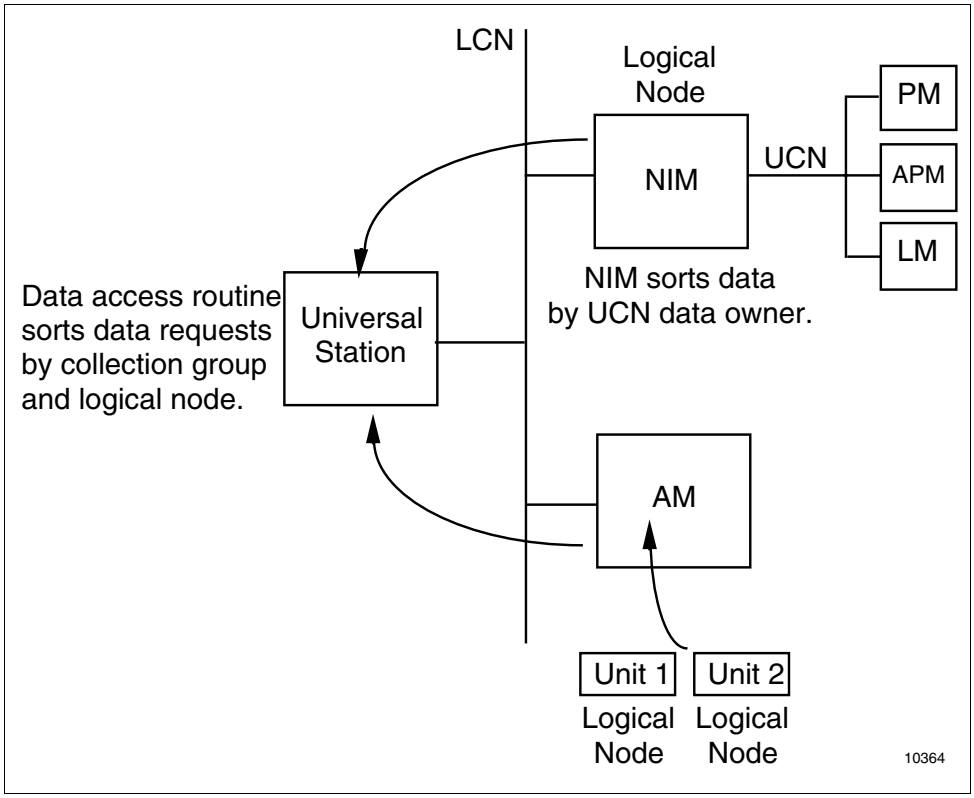


Figure 4 – Data Access of Logical Nodes

Reducing LCN messages

Reducing the number of collection groups per logical node or creating homogeneous collection groups (containing parameters from only one logical node) reduces the number of LCN messages the data owner will have to service.

The data owner handles more parameters per message if the number of parameters per logical node in a given list is increased.

In general, however, more overhead time is required to process additional messages than to handle additional parameter requests in a message.

Variables per Collection Group

A collection group can contain 512 variables; however, because an LCN request can contain only 200 parameters per LCN message, it is not useful to have more than 200 variables per logical node in a single collection group.

Optimize Command

In R530 and later, a Picture Editor “Optimize” command is available for graphics containing UCN point.parameters. This command automatically allocates UCN point.parameters into collection groups based on whether they are control resident or IO Link resident parameters. This command must be executed for each UCN that is to be optimized for a particular custom graphic.

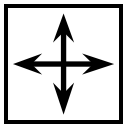
The Optimize command requires three values:

- UCN number
- Collection group number for control resident parameters
- Collection group number for IO link resident parameters

The format of the command is as follows:

OPTIMIZE <UCN> <Control Group> <IOP Group>

When this command is issued, the picture editor loops through all references in the symbol table to determine which parameters are on the specified UCN. The picture editor assigns the parameters of the points in the control processor to the first group, and then assigns the parameters of the points that reside in the IOP hardware to the second group.



REFERENCE—Appendix I of the *Picture Editor Reference Manual* provides additional information about schematic loading and collection sets,
Binder TPS 3032-2.

For your convenience, Appendix I is included at the end of this course module.

Lab Exercise

Optimizing a Custom Display for Data Access

Instructions

In this exercise you will build a custom graphic using UCN parameters and you will optimize it for data collection on the LCN.

1. Select a control resident digital composite and regulatory point from your UCN partition.

2. Detail each point in order to determine its component IOP points and the UCN on which they reside.

3. Build a picture editor display utilizing PV and OP values from all of the control resident and IO link resident points.

4. With the graphic in the picture editor, type in the command
   SET COLLECT

   Result: All parameters in the symbol table are assigned to the default settings of Rate 1 and Group 0.

5. Select CANCEL to return to the main graphic.

6. Now type in the command
   OPTIMIZE <UCN> 100 200
   (where UCN is the network on which the points reside, 100 is the collection group number for the control resident parameters, and 200 is the collection group number for the IO link resident parameters.

7. Now type in the command SET COLLECT and observe how the picture editor has allocated the parameters into collection groups.

Appendix

Schematic Loading

(Excerpt from Picture Editor Reference Manual)

SCHEMATIC LOADING
Appendix I

I.1.1 SCHEMATIC OPTIMIZATION

It is good engineering practice to optimize schematics wherever possible to reduce the load on the TPS Network system. This is particularly true when complex plant operation requires the use of large, sophisticated schematics. It applies equally well to users with less demanding display requirements.

I1.1 General Guidelines to Reduce Schematic Loading

There are several ways to improve schematics, reduce memory requirements, improve performance, and reduce the load on LCN and UCN nodes.

- **Use as few parameters as necessary in any schematic**—This seems obvious, but there may be a considerable difference between the schematic that gets the job done and one that does the job efficiently with the least system impact. You should evaluate the design carefully.

- **Keep the picture as simple as possibl**e—This principle has human factors impact as well as system implications.

- **Static parameters**—set the update rate to zero for any static parameters (those that don't need updating). They are collected only at schematic invocation.

- **Lines**—If possible, connect multiple lines with the same characteristics (behavior, etc.). For example, it is better to draw a box using four connected line segments, rather than four separate lines positioned to look like a box. Connected lines use less memory and take less time to draw on the screen.

- **Text**—If there is text on the same line with the same behavior, it is better to use a text object containing multiple words and spaces than to create separate text objects. For example, it is better to make the text "Boiler Feed Valve" with spaces between words, rather than create separate texts objects "Boiler," "Feed", and "Valve", placed on the same line. It uses less memory and reduces drawing time.

- **Similar objects**—Minimize the number of copies of the same object (for example, a reactor outline) in a schematic. It is easy to copy the object and change minor characteristics, like color, blink, etc. A better approach is to minimize the number of times the same (or very similar) objects are drawn by using conditionals to change color or other characteristics.

- **Variants**—Use Overlays Instead of Variants for target-initiated Changes. Any parameter in a Variant (every limb) is collected continuously as determined by its collection group number and update rate regardless of whether the Variant or limb is used. This is not true for overlays.

  Overlay parameters are only collected when the overlay is on the screen. Therefore, rather than use a Variant with multiple limbs for target-initiated changes, try to use multiple overlays that produce the same effect. Obviously, you should still use Variants when you want changes in conditions to cause behavior changes in the display.

- **Custom Change Zones**—Consider the following when building custom change zones:

  a. A Change Zone is always more efficient if built and used as an overlay. Build the Change Zone as an overlay, then have the actor that invokes the Change Zone call the appropriate overlay.

  b. For fast update parameters in a schematic, build a custom change zone in a small part of the schematic and put the fast update items in this change zone. Put the slowest possible update rate on all other parameters in the picture.

  c. Avoid the USER_CZ actor when it is not necessary because it takes display update time. If you do use USER_CZ, remember that it is unnecessary to do an update. After invoking USER_CZ, the function does its own display update.

- **Multiple use of the same schematic**—Don't build a huge schematic that shows the entire process and all data and then have this same schematic on multiple Operator Stations in the same console. Instead, create an overview schematic that shows the whole process with as few parameters as possible. Then build separate schematics that show smaller parts of the process, with as few duplicate parameters as possible.

- **Elegant displays**—Some graphic capabilities produce elegant displays, but impact display update time and processor usage. These include:

  a. Circles: octagons are as pleasing aesthetically and more efficient.

  b. Scaleable text: this isn't really text but a set of lines and solids that impact performance.

  c. 3-D Graphics: they significantly slow a graphics update, particularly when used with a condition or in a subpicture.

- **Subpictures**—Avoid lines and solids without conditions in subpictures if possible. Subpictures (including text, lines, and solids) are redrawn every update even if there are no conditions, variants, or values. The effort required to include these objects in each picture pays off in update speed.

- **Reverse video**—Use reverse video text and even reverse video spaces instead of highlighting solids. Because the display of text is much faster, you can improve a graphic by replacing boxes with reverse video spaces.

I.2 REDUCING SCHEMATIC LOADING ON ALL NODES

The concepts that follow rely heavily on the assignment of parameter requests to collection groups. If you are not familiar with the concept of a collection group, refer to the Set Collection command in subsection 3.3.1.

I.2.1 Using Longer Update Periods

It is important to reduce the rate of data gathering. The standard update rate for custom schematics is once every 4 seconds, but it is not always necessary to collect data at that rate. Consider collecting data at longer periods, particularly for data that you do not expect to change rapidly. The level of a large tank, or the temperature of a relatively slow process may not change significantly for tens of seconds or even minutes, so there is no need to update values such as these every 4 seconds.

Also, any group of values can be updated on demand using the UPDATE actor, so slow values can be collected when the operator needs them, using appropriate targets. A large class of data (point names, descriptors, engineering units, etc.) do not need to be updated at all. These data can be collected once, at schematic invocation.

I.2.2 Grouping by LCN Node Type and Unit

To further reduce loading, each collection group should contain parameters from a single LCN node. In the case of AM and CG nodes, use separate collection groups for parameters from each Unit accessed by those nodes.

The same principles are true for the NIM, but you need to consider several other types of partitioning. This is discussed in depth later.

I.3 REDUCING SCHEMATIC LOADING ON UCN NODES

There are two concepts that particularly apply when optimizing the UCN data gathering efficiency of a custom schematic. One is grouping of the data access requests by processor type and the other is the use of parameters at the highest possible data owner level.

I.3.1 Grouping of Data Access Requests by Processor Type

Custom schematics request data from the network through LCN Data Access. The mechanism used to make the request is called an Intermediate Data Block (IDB), but in this document, it is called a Data Request message.

Each Data Request message consists of a single collection group. A completely homogeneous request is generated by a group containing parameters from

- a single UCN node, and
- a single processor type (Control or IOL) within that node

However, if this criteria is applied strictly to more than a few UCN nodes or in cases where there are multiple collection rates, it creates a large number of IDBs and places a heavy transaction load on the NIM. It also creates a bookkeeping problem for the developer.

NOTE

The NIM sorts by node so display builders only need to be concerned about grouping by processor type and collection rate.

The combination of user-grouping by processor type and NIM sorting by node creates a UCN request message that is resident-homogeneous (single node/single processor in the same node).

The need for homogeneous UCN Request Messages arises from the way in which parameter requests are processed in the PM or APM. Control parameters are (in effect) obtained directly from memory with relatively small cost in processing time in the node's Communications Processor. IOL parameters require that the Communications Processor create a second request to the IOL processor with an associated cost in processing time.

This overhead cost occurs each time a new IOL parameter is encountered in a message. In a message that consists of a mixture of control-resident and lOL-resident parameters, the Communications Processor uses a significant amount of time switching between the two types of parameters. If the message consists of IOL parameters only, the number of IOL processor requests is considerably reduced because the Communications Processor can include several parameters in a single IOL request.

You can determine the processor type from the point type. It isn't necessary to be aware of the actual residence of a given parameter. The relationship of point type to processor type is—

IOL Processor Point Types

- Digital Input
- Digital Output
- Analog Input
- Analog Output

Control Processor Point Types

- Regulatory Control
- Regulatory PV
- Digital Composite
- Logic
- Process Module

All point types have some parameters that are NlM-resident, but access to those parameters has no effect on UCN node communications loading.

I.3.1.1 The Basic Rules for Grouping

Using the update rate considerations previously discussed, set up the collection groups as follows:

- groups to be updated at target selection
- groups to be collected at multiples of the schematic base rate
- groups to be included in fast update

Split groups that will update into subgroups by processor type (control or IOL).

Example

You are creating a schematic that requests both Control and IOL data from multiple APMs. Some of the data is non-updating (descriptors, point names, etc.).

You determine that the best overall system loading is achieved using two update rates (for example, the standard 4-second update and 8-second update).

Because the numbering of groups is arbitrary within the 0-245 limit allowed by the Picture Editor, you establish the following groups:

Group	Rate	Membership
0	0	All non-updating parameters
10	1	All ACKSTAT collectors
101	1	Control parameters collected every 4 seconds
102	1	IOL parameters collected every 4 seconds
201	2	Control parameters collected every 8 seconds
202	2	IOL parameters collected every 8 seconds

When the NIM constructs UCN parameter requests, it sorts each group's parameters by UCN node and each APM receives a homogeneous request.

I.3.2 Use of Parameters at the Highest Data Owner Level

After determining the basic parameter requirements of an object (Variant, Conditional Behavior, etc.) you should ask the following questions if the logic requires PM/APM parameters:

- Is there a NlM-resident parameter that will allow creation of an equivalent effect?
- If the answer is no and if the parameter is lOL resident, can you use a parameter that is resident in the Control Processor?

For example, you may need a Digital Input point's PV that is already in use as a parameter of a logic point or digital composite point. In that event, the schematic can request a control-resident parameter, with less effect on Communications CPU loading.

Consider the following schematic Variant:

```
IF (A100.PVHHFL) THEN SUBPICTURE HIHIALRM ELSE
IF (A1 00.PVHIFL) THEN SUBPICTURE Hl ALARM ELSE
IF (A100.PVLOFL) THEN SUBPICTURE LO ALARM ELSE
  SUBPICTURE OTHRALRM
```

Assuming that A100 is an IOL resident point and that the parameters are in the same collection group, three separate transactions are required:

- the data request message (IDB) from the US to the NIM
- a UCN parameter request from the NIM to the APM which owns A100
- an IOL request generated by the host APMs Communications Processor.

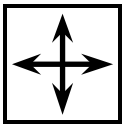
Now consider the following schematic Variant, with basically the same effect:

```
IF (A100.HIGHAL=PVHH) THEN SUBPICTURE HIHIALRM ELSE
IF (A100.HIGHAL=PVHI) THEN SUBPICTURE Hl ALARM ELSE
IF (A100.HIGHAL=PVLO) THEN SUBPICTURE LO ALARM ELSE
  SUBPICTURE OTHRALRM
```

The data collection is reduced to a single NlM-resident parameter, HIGHAL, which is collected by a single transaction (a data request to the NIM).

This is a rather simple illustration of using parameters at the highest data owner level possible.

Directions



DIRECTIONS—This is the end of the study material for this module.

LAST PAGE