Honeywell

CM50S User Manual

CM11-430

Implementation
CM50S

CM50S User Manual

CM11-430
Release 4.1
7/93

Copyright, Trademarks, and Notices

Revision 02 – July 1, 1993

About This Publication

This publication provides information on the writing, installation, testing, and modification of Application Programs that execute in the CM50S (Rel 4.1) environment as an integral part of a TDC 3000 Process Control System. It does not, by itself, cover everything you need to know to develop advanced control schemes. Essential topics covered elsewhere include

- TDC 3000 System Concepts
- TDC 3000 System Configuration
- FORTRAN, "C," and Pascal Programming Languages

A list of related publications is located in paragraph 1.3 of this publication.

Users of previous CM50S releases will find a discussion of support for vintage interface routines at paragraph 1.4

You should be aware that all programmatic interface calls to CM50S are grouped by programming language. Tab 2 precedes the FORTRAN calls (sections 9 thru 12), Tab 3 precedes the Pascal calls (sections 13 thru 16), and Tab 4 precedes the C language calls (sections 17 thru 20).

NOTE

This publication supports Release 4.0 and 4.1 of CM50S.

NOTE

Change bars in this publication are used to indicate the technical changes that were made at Release 4.1

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

INTRODUCTION
Section 1

This section discusses uses for a DEC VAX in a TDC 3000 system; reviews the most significant hardware and software components of CM50S; and lists the other publications you need to consult during implementation and operation of CM50S.

1.1 DEC VAX INTERFACE TO TDC 3000

By the addition of CM50S hardware and software, your DEC VAX can become a fully integrated node in a TDC 3000 control system, thus enabling it to exchange information with all other nodes on the same LCN. At the same time, the DEC VAX communications capabilities enable it to serve as a data-gathering node within an overall plant-wide management information system. This combination of capabilities provides a broad range of potential applications.

Among the expected advanced applications for a DEC VAX in a TDC 3000 system are

- Custom Report Generation
- Long Term Data Storage
- Process Scheduling
- Process Optimization
- Plant Management Tasks

1.2 DEC VAX INTERFACE ARCHITECTURE

The DEC VAX interface to TDC 3000 consists of three pieces:

- A DEC VAX or MicroVAX computer with VAX/VMS (Release 5.4.1 or later), the FMS Runtime Package and, if connected to a PLNM, the LAT Master Package and LAT Patch CSC 511.
- Either a Computer Gateway (CG) with an HDLC connection (and a CM50S communications board installed in the VAX) or a Plant Network Module (PLNM) connected to the VAX by the Local Area Terminal (LAT) protocol.
- CM50S software from Honeywell IAC. This CM50S software consists of IAC-provided extensions to VMS that handle the details of communication between user-written application programs in the VAX and other LCN nodes.

See Figure 1-1 for an overview of the DEC VAX interface architecture. (Note that CM50S can support either up-to four CGs or up-to four PLNMs and that each CG/PLNM can be connected to the same or to different LCNs.)

The CG and PLNM are standard LCN nodes. The CG hardware components include an LCN interface board, an MCPU, a memory, a power supply, and a communication interface board for an HDLC connection link to the VAX. Its memory contains the standard TDC 3000 node-environment software along with CG-specific application software and a user-defined database loaded from the LCN.

The PLNM differs from the CG by the replacement of the HDLC interface with a Computer Network Interface (CNI) board for the DECnet connection. The CNI board is downloaded with communications software from the VAX.

The CG/PLNM database includes two data point types that are used for controlling the scheduling of user programs in the VAX and for holding data to be exchanged between the VAX and other nodes on the TDC 3000 Local Control Network.

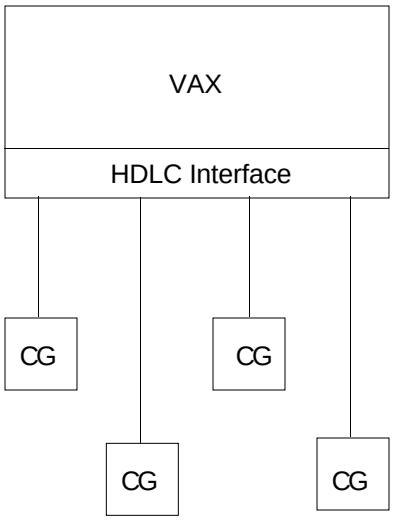
1.3 REFERENCES

The following TDC 3000 publications contain additional information required to understand functions of the CM50S package. For VAX/VMS-related questions, refer to the appropriate DEC manuals.

1.3.1 Publications Found in the TDC 3000 LCN Bookset

Title	Binder
Network Form Instructions	Implementation/Startup & Reconfiguration - 1
System Control Functions	Implementation/Startup & Reconfiguration - 2
Computer Gateway Forms	Implementation/Configuration Forms
Computer Gateway Form Instructions	Implementation/CM50S
Computer Gateway Parameter Reference Dictionary	Implementation/CM50S
Data Entity Builder Manual	Implementation/Engineering Operations - 1
Picture Editor Reference Manual	Implementation/Engineering Operations - 2
Hiway Gateway Parameter Reference Dictionary	Implementation/Hiway Gateway - 1
Hiway Gateway Control Functions	Implementation/Hiway Gateway - 1
Process Manager Parameter Reference Dictionary	Implementation/Process Manager - 2
Process Manager Control Functions and Algorithms	Implementation/Process Manager - 2
Application Module Parameter Reference Dictionary	Implementation/Application Module - 1
Control Language/Application Module Reference Manual	Implementation/Application Module - 2
Control Language/Application Module Data Entry	Implementation/Application Module - 2

HDLC Interface



LAT Interface

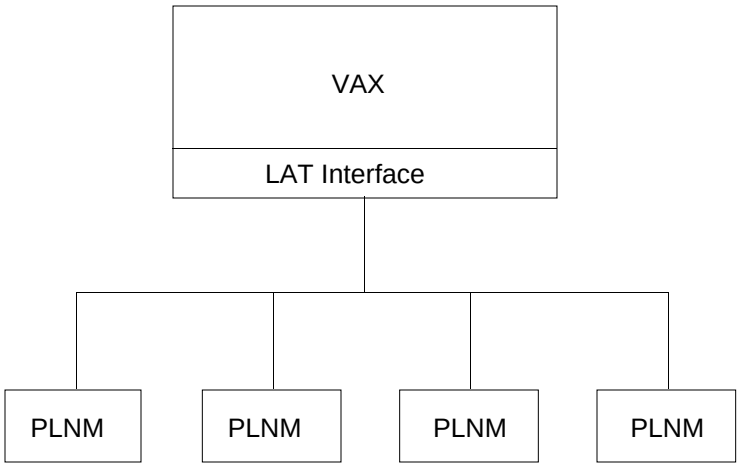


Figure 1-1 — Alternative CM50S Hardware Organizations 4664

1.4 SUPPORT FOR VINTAGE INTERFACE ROUTINES

Improvements in user-interface routines since the first release have allowed CM50S to become a more integrated and user-friendly product. In cases where additional options are defined for future TDC 3000 releases, the interface routines allocate space for those options to ease the migration path. For the convenience of existing CM50S users, the old procedure calls (both Pascal and FORTRAN) continue to be supported to allow you to use existing programs without change; however, Honeywell IAC strongly recommends converting existing programs to use of the new interface calls to simplify future system maintenance and to take advantage of new capabilities.

See Appendix G—Vintage Procedures—for a complete reiteration of the old user-interface procedure calls.

1.5 RESTRICTIONS ON ENTITY NAME LENGTHS

CM50S internally supports 16-character ACIDP names and 20-character LCN entities (a 16-character point name that can be preceded by an optional "pinid" prefix for Network Gateway routing) for all current calls. However, not all LCNs will recognize point names longer than eight characters.

Any LCN running under TDC 3000 Release 320 or earlier will receive only the first eight characters of the point name from the VAX. Any LCN running under TDC 3000 Release 400 or later will receive the full-length entity names, even if the system-wide values on that LCN limit point names to eight characters. If an entity name greater than eight characters is sent to a Release 320 or earlier, it is truncated (so as to match an LCN name with the first eight characters). If the same entity name is sent to a Release 400 or later LCN that has only short point names, it will cause a "point not found" return.

All point name arguments used in CM50S calls must be declared to be their full length and have values that are either filled with trailing blanks (FORTRAN & Pascal standard) or are terminated with a null character ("C" standard).

The earlier CM50 calls that used eight-character point names are still supported as vintage calls. The vintage calls may be used with any release of TDC 3000 as long as the actual point name being addressed contains no more than eight non-blank characters.

CONCEPTS AND MECHANISMS
Section 2

This section introduces you to the use of the software features that have been added to a VAX computer to provide a simple, efficient interface to data in devices on the TDC 3000 Local Control Network.

CM50S serves as a platform for computer applications connected to one to four LCNs. It provides routines for activation of programs, moving sets of data values between the LCN and the host computer, and for managing those programs and data sets.

2.1 EXCHANGING DATA WITH AN LCN

Before starting an explanation of how application programs get and give LCN data, a short explanation of the TDC 3000 database organization and data access is appropriate.

Because the TDC 3000 is a distributed system, its data is spread among the various nodes. Each piece of data is assigned to a data owner (program) in the node where it resides. To address a specific data item on the LCN, you provide character strings that identify both a point name and the name of a parameter applicable to that point type (referred to together as **point.parameter**). This information is used in a request that is broadcast to all nodes on the LCN. The data owner responds with a numeric "internal address" that can be used for direct access to the data. When a program requests a piece of data by point.parameter name, the "internal address" must be obtained each time. Time and LCN loading is saved by having the program obtain the "internal address" and saving it for use each time the data is requested.

Because CM50S can be connected to up-to-four Computer Gateways or Plant Network Modules (on the same or different LCNs), data access requests must specify which CG/PLNM is being addressed. Within the VAX, CG/PLNMs are addressed by a VAX port number (not by its LCN node number).

Several sets of interface routines are provided to enable access to process values held by data points on the LCN. Your choice of which type of Get/Store interface routine to use is affected by characteristics of the data to be processed.

For example, to access one parameter (or one element of a parameter array) of a single data point, you would use a single-point interface routine. To get the same type of single-value data for multiple points, you could use either a Multipoint or a Point List request. On the other hand, if you need an efficient way to get all the data in a parameter array, you could use a single-point or a Raw Data request.

- **Get/Store Current Process Values:** Current process values are obtained from the CG/PLNM, the Applications Module, the Hiway Gateway (data from Hiway boxes), or the Network Interface Manager (data from UCN boxes). The value of any point.parameter on the LCN may be read by CM50S. Storing of values from CM50S to the LCN is subject to restrictions established by LCN configuration choices and by the control mode in effect for the destination point.

- **Get History Values:** History-value accumulations stored on the History Module (HM) may be retrieved into CM50S as either snapshots or averages. Averages are available for periods ranging from a configurable minimum of 3 to 30 minutes to a maximum of a month. For each average, the minimum and maximum values and the number of valid snapshots collected during the period are also returned.

- **Get/Send Messages:** A control program can send character-string messages to all operator stations within its operating area by using the Send Message interface routine. An option to wait for operator confirmation is provided. Other devices on the LCN can send character-string messages to CM50S. These messages are received by the CG/PLNM and held, pending the control program's call of the Get Message interface routine. Presence of a pending message at the CG/PLNM is detected by the Get Status interface routine.

2.1.1 Single Point Transfers

Single-point calls get or store a value for one point.parameter per request. For array parameters, the request can be for the entire array or for a specific array element (by its index value). Note: Although single point transfers are generally less efficient than multi-point transfers, they are the most efficient mechanism for transferring arrays—since the multi-point calls request each element of an array as a distinct value.

For the special case of transferring Real values between LCNs through CM50S, a special set of single point calls bypasses the normal conversions between LCN and VAX floating-point formats. These raw data requests get or store all values of a parameter array in unconverted (LCN) format, one point for each request. Raw data requests work only with arrays of data type Real.

2.1.2 Data Definition Tables

Multipoint requests get or store data for a single parameter (or parameter array element) for each specified point. (You could use this type of request to process a parameter array, but it would require a separate entry for each individual array element.) This type of request is controlled by a Data Definition Table (DDT). The DDT specifies the set of point.parameter values to be accessed and how each value is to be processed.

DDT tables are of three distinct types: Input, Output, and History. Within an Input or Output table the points are grouped by data type (real, integer, ASCII, enumeration, etc.). Individual point and parameter names are specified along with individual point-value processing information.

A Data Definition Table provides a list of up to 300 tagnames (of a maximum of 4 different data types) whose values are to be transferred as a synchronized unit. All points specified in a DDT must be on the same LCN (i.e. addressable through the same CG/PLNM. A DDT may also define data filters ("table processing") that are to be applied by CM50S. For DDTs, the validation of the point.parameters is done ahead of time so that the run-time calls reference the values efficiently by their internal addresses. Note that the DDT is not part of the calling program; it is separately constructed and installed.

The use of DDTs is supported by a comprehensive set of routines for managing multi-point data transfers, but does involve some disk overhead at run time. This run-time overhead can be avoided by using the Multi-Point List routines.

A DDT can be referenced by more than one application program, but this practice is not recommended. If the DDT uses the option of updating a values file, the concurrent use of that DDT by multiple processes is not permitted.

2.1.2.1 DDT Management

DDTs are generated from source text files that specify the points and parameters to be transferred along with any option Table Processing operations. It is recommended that users build and revise DDT source files through the Edit screens of the DDT operations interface, as this interface will assure that all the syntax requirements are met.

When a DDT is built from a source file, its characteristics are recorded in the CM50S global common and a set of disk files are created to support the management of DDT data transfers. The DDT management utilities include reports of the status of all built DDTs and detailed information about any specific DDT along with options to change the characteristics or delete a DDT or associate it with a specific control program and ACIDP.

The DDT management functions may be executed through the forms-driven terminal interface, DCL commands or programmatic calls within the VAX.

2.1.2.2 Table Processing Options

Value transformations for input and output DDTs are performed by the Table Processor. The data filtering options are:

- Test mode value substitution (input only)
- Bad Value substitution (input only)
- Value scaling (for Real values, a choice of 9 transformation algorithms)
- Limit checking
- Value clamping
- Source/Destination reordering of data arrays

Table processing also allows for the values (both before and after filtering) for the most recent transfer to be recorded in a disk file. This Value Table option allows complete monitoring of the data transfer process, but can have a significant performance effect on a system with heavy disk utilization.

Run-time options allow multi-point data transfers to bypass the table processing operations and place the data directly into the appropriate data arrays specified by the call.

2.1.3 Multi-Point List Transfers

A Multi-Point List (MPL) is a memory-resident list of internal identifiers of a group of up to 300 tags. An MPL can contain tags of up to four different data types. The structure of a Multi-Point List is identical to the structure used by the CG/PLNM to process DDT requests.

The CM50S MPL interface routines use memory-resident MPL definitions (instead of reading DDT disk files at run time). Thus, performance conscious users can improve throughput by using MPLs to bypass the management overhead of DDTs.

MPLs offer increased flexibility over DDTs in that the same MPL can be used for both input and output, multiple users can make concurrent data requests using the same MPL, and an MPL request can be redirected through a different CG/PLNM at runtime. There is no limit to the number of MPLs that can be generated. On the other hand, the Table Processing options and DDT management utilities are not available to MPLs.

The Multi-Point List routines support getting and storing current process values, retrieving history data, generating MPLs for lists of tagnames, and saving/reading MPLs to/from disk. Note: Any DDT may be used by the MPL routines by reading its Internal Identifiers file (.II).

2.1.4 Point List Transfers

Point List data requests get or store data of a specified type from/to groups of points without the use of DDTs. Each entry in a Point List request affects a single point.parameter or a single element of a parameter array. Each call may transfer up to 300 values of the same data type (real, integer, ASCII, enumeration, time, point_id, etc.)

2.1.5 Dynamic Indirection

Some LCN applications use dynamic indirection to reference a tag whose Entity Id is stored in a custom data segment. A program in the VAX can follow this indirection by making a pair of Single Point calls. The first call retrieves the Entity Id (using value type = 15) from the custom data segment. The returned value then is used as the entity name in a Single Point Get or Store (External Id) function call to access the referenced value.

2.2 USER PROGRAM CHARACTERISTICS

User-written application programs supported by CM50S execute within the environment provided by the VMS operating system, but also uses special interface routines that simplify the tasks required in the exchange of data with other TDC 3000 nodes and boxes. These interface routines are compatible with your use of FORTRAN, Pascal, or "C" programming languages. Specifics of interfacing routines in each of these programming languages are found following Tabs 2, 3, and 4 respectively.

Each CM50 application program falls into one of the following categories:

- Data Acquisition Programs (DAPs)—use a one-way interface to retrieve values from the LCN for analysis and storage in the VAX but have no affect on any control operations.

- Advanced Control Programs (ACPs)—provide a two-way interface for exchanging data between the VAX and the LCN and thus need to be more tightly bound to process control operations.

- Indirect Control Programs—use "surrogate" ACPs that enable them to store data in LCN points without meeting all the requirements of an ACP.

Each program normally includes three stages, each with its related interface routines.

Setup stage—The first executable statement of an ACP should call the ACP Initialization-interface routine (CM50_SET_ACP) to map the ACIDP connection and establish the CM50 exit handler. Use of this interface routine is optional for DAPs and Indirect control programs.

If the program can be activated from the LCN, the Get ACP Status-interface routine (GETSTS) should be used to obtain information that indicates why the ACP has been activated (and clear the pending-request flags), thereby establishing what actions may be necessary at this specific activation.

Run stage—The main body of the program handles the desired data manipulation and logical processing. Routines for exchange of data with LCN-resident modules are available to get point data, store point data, receive messages, send messages, and get history data.

Also available are bad-data handling routines, an external to internal address-conversion routine, LCN time to external time conversions, a program delay routine, and a program hibernate routine.

Cleanup stage— If the program is subject to demand execution from the LCN, then program termination should be preceded by another use of the Get ACP Status interface routine to determine if an additional processing cycle has been requested while the program was running.

As a final call, the ACP Termination routine (PRGTRM) provides status information for trap handling by the CM50S exit handler. Use of this interface routine is optional for DAPs and Indirect control programs).

2.2.1 Data Acquisition Programs (DAPs)

A Data Acquisition Program retrieves data from LCNs for processing in the host computer. A DAP may retrieve data from multiple LCNs. A DAP is not connected to an ACIDP and does not have any affect on process control operations. (Therefore use of the CM50S initialization and termination routines is optional.) Typical applications of Data Acquisition Programs are:

- Long term history collection and periodic reporting, where the DAP is activated in the host on a regular schedule to retrieve data from the LCN and store it in a database or format it into a printed report. These programs generally run as detached processes, have no direct user interaction, and use MPL or Point List calls to retrieve a large number of values during each processing cycle.

- On-line analysis, where the DAP is activated interactively from a user terminal to monitor and/or perform analytic calculations on current process conditions. These programs may be executed concurrently by several users, may use a graphics interface (such as X windows), and are apt to use single point calls for ad hoc selection of points to retrieve or MPLs for predefined data sets.

- Event capture, which is similar to long term history collection except that the DAP is activated based on an asynchronous event or computer operator demand. (In order to detect LCN events, an application program must be installed as an ACP.)

2.2.1.1 General Restrictions

- A DAP can only Get point data, not Store it.

- A DAP cannot use the Get Message or Send Message interfaces.

- A DAP cannot be activated from the LCN.

2.2.1.2 DAP Activation

Data acquisition programs are activated using the standard VMS facilities, such as the RUN command. Periodic scheduling of a DAP is normally done using the CM50 Scheduler utility.

2.2.1.3 DAP Trap Handling

Use of the ACP Initialization routine by a DAP is optional, but recommended.

2.2.1.4 DAP Program Suspension

A DAP can use the ACP Delay function to suspend its operation for up to 60 seconds.

2.2.1.5 DAP Termination

Use of the ACP Termination function by a DAP is only used (and then is required) when it has used the ACP Initialization routine.

2.2.2 Advanced Control Programs (ACPs)

Any program that directly stores values to an LCN or must react to events on the LCN is considered to be an Advanced Control Program (ACP). These programs are tightly bound to a specific LCN through an ACIDP (see heading 2.3). An ACP can retrieve data from multiple LCNs, but is allowed to store data only through the CG/PLNM containing its associated ACIDP. Typical ACP applications are:

- Information download—to obtain economic and management data from VAX files and/or data from laboratory instruments or other LCNs and stores it in custom data segments on an LCN for use on operator displays. In the event of VAX or link failures, these displays continue to show the most recently stored values.

- Recipe/Target update—to perform analytical calculations in the VAX to optimize recipe or target values used by CL programs for process control. These programs rely on the CL code (in an AM or other device) to confirm the success of the download and to respond to failures with appropriate control shedding.

- Direct control—to store calculated values to set points or other parameters that have a direct effect on the control process. These programs should be connected to an ACIDP that is configured with Remote Cascade Enabled so that control will be shed appropriately in the event that communication fails or the CM50S is shut down.

2.2.2.1 General Restrictions

- Each ACP must be connected to an ACIDP, therefore a maximum of 250 ACPs may be installed per CG/PLNM.

- An ACP cannot be executed concurrently by multiple users.

- An ACP must include the setup and cleanup stages

- An ACP does not require user interaction.

- The standard system input and output files for an ACP must be simple text files (no graphics interface). By default, the standard SYS$OUTPUT is directed to the NULL device.

2.2.2.2 ACP Activation

An ACP normally is initiated from the LCN through its associated ACIDP. Scheduling by an ACIDP can be configured as time-based (periodic or cyclic), operator demand initiated, or a combination. The ACP also is activated when the the ACIDP's Point Process Special (PPS) flag is set by an Event Initiated Point (EIP) or a programmatic store from another LCN node.

An ACP can be activated from the VAX using the CM50S interface. The ACP activation routine may be invoked through the ACP Operations forms, a DCL command or programmatically. The standard VMS program activation facilities (such as RUN) do NOT provide the ACIDP synchronization required for advanced control applications.

If sophisticated scheduling is required (for example, activating the ACP at different times depending on the day of the week), then the ACP Activation command may be placed under the CM50S scheduler utility.

2.2.2.3 ACP Trap Handling

Use of the ACP initialization routine is required.

2.2.2.4 ACP Suspension

An ACP can use the ACP Delay function to suspend its operation for up-to-60 seconds and/or use the ACP Hibernate function to suspend its operation until the next external activation (EIP, operator demand, or schedule).

2.2.2.5 ACP Termination

Use of the ACP Termination function is required.

2.2.2.6 ACP Status Table

The connections between Advanced Control Programs and ACIDPs are maintained in the ACP Status Table.

Programs are installed as ACPs, connected to ACIDPs and deleted from this table using the ACP Operations utility (or the ACP DCL command or programmatic calls). All of the VMS options (standard file assignments, process name, priority, special privileges and quotas) used to run the ACP as a detached process are specified when the ACP is installed and are maintained in this table.

2.2.2.7 ACP Installation Modes

Data transfer by an ACP is affected by the program's installation mode, as specified in the ACP status table.

- **Normal Mode Operation:** When a program is installed as an ACP in Normal mode, all forms of ACP activation are permitted and live data values are used in the data transfers. Acceptance of stores to regulatory parameters are controlled by the current mode of the destination point on the LCN and Data Hiway or UCN.
- **Restricted Mode Operation:** When a program is installed as an ACP in Restricted mode, any attempts to store data or to send messages are blocked. Time-based scheduling is replaced by ACP Operations screen activation. Programs cannot be disconnected from their corresponding ACIDPs while in Restricted Mode. An ACP from ACIDP disconnect can be done only after a change of mode to Normal or Test.
- **Test Mode Operation:** Test mode operation is the most restrictive mode of ACP operation. All store or send message requests are blocked. Test values are substituted for live input data when provided for in the DDT Get Data request's Data Definition Table (no test value substitution is done for non-DDT data transfers). Time-based activation of any ACP in test mode is disabled.

2.2.2.8 ACP Execution States

An installed ACP will be in one of the following Execution states:

- **ABORT**—The program has aborted and the ACIDP must be cleared from the US, or by disconnecting it from the ACP and then reconnecting it, before the program can be triggered again.

- **ACCESS**—The ACIDP is controlling an active LCN Data Access request to transfer values. The ACP program is suspended while waiting completion of the I/O request.

- **DELAY**—The program is not running in the VAX, but it is scheduled to be triggered by its ACIDP.

- **FAIL**—The program aborted in the VAX without sending an abort message to the LCN.

- **HIBR**—The program is resident in the VAX, waiting for a trigger before it resumes processing. Note that this state is visible only on the VAX; the ACIDP shows RUN state on the LCN.

- **OFF**—The program is not running in the VAX, but can be activated normally.

- **PEND**—The program has been spawned by the VAX but has not started execution. This state should be very transitive. If a program remains in this state it means that either
  1) the program does not call CM50_SET_ACP (or ACPTRP) as its first statement; or
  2) the VAX priorities are such that the program never gets a chance to execute; or
  3) a VAX access control problem has occurred when opening SYS$INPUT, SYS$OUTPUT, or SYS$ERROR and the process has been killed without executing the program.

  Note that this state is visible only on the VAX; the ACIDP shows RUN state on the LCN.

- **RUN**—The program is currently running.

- **WAIT**—The program has suspended itself using the CM50_ACPDELAY function. It will resume running in from 1 to 60 seconds. Note that this state is visible only on the VAX; the ACIDP shows RUN state on the LCN.

2.2.3 Indirect Control Programs

Some applications must store values to an LCN but do not meet the restrictions imposed on an ACP. Examples are: storing values to more than one LCN, multiple concurrent users with occasional writes to an LCN, or a graphics program with an option to send calculated results to an LCN. These applications can be implemented by writing Indirect Control Programs that use ACIDPs connected to Surrogate ACPs.

A Surrogate ACP is a program that does no real processing but is connected to an ACIDP for the sole purpose of allowing other programs to communicate to the LCN through its ACIDP connection. The surrogate ACP spends most of its time suspended, leaving its ACIDP in RUN state. Deactivating the ACP will block stores to the LCN for all Indirect Control Programs using that ACIDP.

An Indirect Control Program behaves exactly like a Data Acquisition Program except that it stores values to an LCN by specifying an ACIDP name in the data transfer request. Note that while an ACIDP is servicing a data transfer request from one user, it shifts from RUN to ACCESS state thereby blocking concurrent transactions from other users. Since successful completion of the store data request is dependent on the current state of the ACIDP, the use of retry logic is recommended.

2.2.3.1 General Restrictions

- An Indirect Control Program needs a surrogate ACP connected to an ACIDP in order for it to store data on the LCN.
- An Indirect Control Program can use the MPL and/or Point List routines to store data on the LCN, but cannot use DDT or Single Point calls.
- An Indirect Control program in a multi-user environment must include retry logic to handle ACIDP contention situations.

2.2.3.2 Indirect Control Program Activation

Indirect Control programs are activated using the standard VMS facilities, such as the RUN command. Periodic scheduling normally is done using the CM50 Scheduler utility.

2.2.3.3 Indirect Control Program Trap Handling

Use of the ACP initialization routine by Indirect Control Programs is optional, but recommended.

2.2.3.4 Indirect Control Program Suspension

An Indirect Control Program can use the ACP Delay function to suspend its operation for up-to-60 seconds.

2.2.3.5 Indirect Control Program Termination

Use of the ACP Termination function by an Indirect Control program is only used (and then is required) when it has used the ACP Initialization routine.

2.2.4 Program Execution Control

Programs that access data on an LCN need stronger controls on their execution states than the normal procedures provided by VMS. CM50S includes routines that maintain the necessary synchronization between processes running on the VAX and the status information kept in the CG/PLNM.

2.2.4.1 Trap Handling

The CM50S trap handler updates the ACIDP status and ACP Status Table during the VMS wrap-up processing which is invoked by both normal and abnormal terminations (but not by the VMS STOP/ID= command or Delete_Process system service). This trap handler is connected to an ACP by the CM50_SET_ACP function (or the vintage ACPTRP procedure). The CM50_SET_ACP function allows the user to specify whether the ACIDP status is set to ABORT (blocking reactivation) or OFF/DELAY in the event of abnormal program termination.

This function also informs the system that the process is using the CM50S shared image, so it is recommended the CM50_SET_ACP be the first executable statement in every program that accesses the LCN.

NOTE

To ensure that ACP terminations are handled correctly and that VMS CM50S data structures reflect the correct termination status, a call to establish the Trap Handler **must** be the first operating instruction of each ACP.

When an ACP terminates, the CG/PLNM is informed that the ACP has gone from "Run" to "Off/Delay" state or to "Abort" state. Once in "Abort" state, the CG/PLNM suspends periodic or cyclic scheduling of the ACP. Reactivation of the ACP requires Universal Station operator demand or a CM50S restart. A program abort alarm is sent to the Real-Time Journal whenever an ACP terminates in "Abort" state.

- **Normal Termination:** Under most circumstances, the CG/PLNM is informed of program completion by a call to the ACP Termination routine (PRGTRM).
- **VMS-Induced Termination:** If VMS terminates an ACP because of a detected error (e.g., an I/O error or divide by zero) an abort code of VMSF is stored in the ACIDP's ABORTCOD parameter.

- **ACP Deactivate/Terminate Request:** The ACP Operations screen Deactivate/Terminate function halts an ACP in either "Off/Delay" or "Abort" state as specified by operator entry.

- **VMS Stop/Identifier Operation:** Use of this privileged VMS DCL function to terminate an ACP bypasses the normal CM50S exit handling. This results in both CG/PLNM and CM50S data structures erroneously showing the ACP to still be in the "Run" state. This problem can be corrected by invoking the ACP Operations screen Deactivate/Terminate function to inform CM50S and the CG/PLNM that the ACP has been terminated.

2.2.4.2 Suspending Execution

- **Program Delay:** A running application program can suspend its operation for a period of from 1 to 60 seconds. During that time, ACP schedule status is unaffected by activities of its ACIDP or by ACP Operations screen activation. (The associated ACIDP remains in RUN state during the suspension.)

- **Program Hibernate:** A running ACP can request suspension until it is reactivated by a CM50S turn-on request from either its ACIDP or the VAX. This allows the ACP to remain memory-resident so that it can respond quickly to a triggering event at the CG/PLNM. When an ACP goes into Hibernate state, the next time it is activated, VMS will not need to create a new process but can immediately wake-up the existing ACP. The ACP will resume processing with the first statement following the call to Hibernate.

2.2.4.3 Trigger Resolution

The Get ACP Status routine retrieves the trigger parameters from the ACIDP, allowing the program to determine whether it was activated based on the CG/PLNM scheduler, operator demand, process demand (EIP), or CG/PLNM initialization. In addition to reading in the current values, this routine resets all of the pending triggers to OFF, so this call can be used to prevent excessive reactivation of an ACP.

2.2.5 Summary of Program Execution Options

The following table summarizes the data exchange and program activation options for the different types of CM50S application programs.

Table 2-1 — CM50S Program Attributes

ATTRIBUTE	PROGRAM TYPE				
	ACP Normal Mode	ACP Rstrct Mode	ACP Test Mode	Indirect Control Program	DAP
Interface routines effective?					
- DDT Get Data	yes	yes	test*	yes	yes
- DDT Store Data	yes	no	no	no	no
- MPL Get Data	yes	yes	yes	yes	yes
- MPL Store Data	yes	acidp**	acidp**	yes	no
- Single Point Get Data	yes	yes	yes	yes	yes
- Single Point Store Data	yes	no	no	no	no
- Point List Get Data	yes	yes	yes	yes	yes
- Point List Store Data	yes	acidp**	acidp**	yes	no
- Raw Data Get	yes	yes	yes	yes	yes
- Raw Data Store	yes	no	no	no	no
- DDT Get History	yes	yes	yes	yes	yes
- MPL Get History	yes	yes	yes	yes	yes
- Single Point Get History	yes	yes	yes	yes	yes
- Get Message	yes	yes	yes	no	no
- Send Message	yes	no	no	yes	no
- ACP Initialization	must	must	must	option	option
- Get ACP Status	yes	yes	yes	no	no
- ACP Delay	yes	yes	yes	yes	yes
- ACP Hibernate	yes	yes	yes	no	no
- ACP Terminate	must	must	must	option	option
- Check Bad Value	yes	yes	yes	yes	yes
- Set Bad Value	yes	yes	yes	yes	yes
- Convert Ext to Int id	yes	yes	yes	yes	yes
- Convert Raw Data	yes	yes	yes	yes	yes
- Get LCN Clock Value	yes	yes	yes	yes	yes
- Convert LCN Time	yes	yes	yes	yes	yes
ACIDP activation?					
- Cyclic	yes	no	no	no	no
- Periodic	yes	no	no	no	no
- Operator Demand	yes	yes	yes	no	no
- Process Special	yes	yes	yes	no	no
- Message Waiting at CG/PLNM	yes	yes	yes	no	no
VAX activation?					
- ACP Operations screen	yes	yes	yes	no	no
- Programmatic Activate	yes	yes	yes	no	no
- VMS RUN command	no	no	no	yes	yes

*Test values specified in the DDT are substituted for live input data .

** If an ACIDP is explicitly named in the call, the success of the store depends on the Access_Key and Execution_State of that ACIDP. If the ACIDP name is defaulted, placing the ACP in Restricted or Test mode blocks the store.

2.2.6 ACP Execution Example

The following is an example of the processing flow during ACP execution. The example is based on activation by an operator on the LCN.

AT A UNIVERSAL STATION

1. The ACP is demanded through its ACIDP's Detail Display

IN THE CG/PLNM

2. The CG/PLNM Scheduler program creates and sends a message requesting the CG/PLNM Communication Handler to turn on the ACP. Data collection from the LCN starts if DDT Prefetch is triggered (the DDT must be CG/PLNM-resident and attached to the ACIDP).

3. The CG/PLNM Communication Handler sends the turn on message to the VAX Communication Handler.

IN THE VAX

4. The VAX Communication Handler sends the turn on message to the VAX Dispatcher.

5. The VAX Dispatcher creates a detached process (unless the ACP is memory resident in Hibernate state) then sends a message to the CG/PLNM to set the ACIDP to RUN state, updates the ACP Status table and wakes up the ACP.

6. VMS opens the preassigned files (SYS$INPUT & SYS$OUTPUT) then the ACP calls the CM50S trap handler as its first statement.

7. The ACP calls the trigger resolution routine to retrieve (and clear) the current values of the ACIDP status parameters and determines how it was activated.

8. A call to a Get Data routine creates a message based on the DDT referenced and sends it to the VAX Communication Handler. The message contents include

   - Sender's Process identification
   - Identification of the CG/PLNM receiver
   - Message length
   - Internal ID and parameter file if needed

9. The ACP task is suspended.

10. The VAX Communication Handler sends the message to the CG/PLNM Communication Handler.

IN THE CG/PLNM

11. The CG/PLNM Communication Handler issues LCN Data Access calls to obtain the value for each tag from its Data Owner node (unless the data was already gathered by a Prefetch trigger).

12. When all the values have been returned and formatted into a return message, the CG/PLNM Communication Handler sends it to the VAX Communication Handler.

IN THE VAX

13. The VAX Communication Handler "wakes up" the suspended task and the interface routine writes the retrieved information into storage allocated by the ACP.

14. The ACP performs calculations on the retrieved data, placing the calculated results in an array.

15. A call to a Store Data routine creates a message based on the DDT referenced and sends it to the VAX Communication Handler.

16. The ACP task is suspended.

17. The VAX Communication Handler sends the message to the CG/PLNM Communication Handler.

IN THE CG/PLNM

18. The CG/PLNM Communication Handler issues LCN Data Access calls to write the value for each tag to its Data Owner node.

19. When the completion status of all the writes have been returned and formatted into a return message, the CG/PLNM Communication Handler sends it to the VAX Communication Handler.

IN THE VAX

20. The VAX Communication Handler "wakes up" the suspended task and the interface routine writes the retrieved status information into storage allocated by the ACP.

21. The ACP checks the success of the store. In case of error, it takes appropriate action, such as sending a message to an operator or setting the termination code to an ABORT value to prevent additional processing until the error condition has been corrected.

22. Upon successful completion of processing, the ACP calls the trigger resolution routine to retrieve (and clear) the current values of the ACIDP status parameters. If another processing cycle has been triggered, it jumps back to step 8.

23. The ACP calls the program termination routine indicating normal completion. (Alternately it calls the CM50_HIBER routine to remain memory resident. This call is followed by a jump back to step 7, which will be executed when the ACP is reactivated.)

24. The CM50S exit handler sends a message to the CG/PLNM to reset the ACIDP to OFF/DELAY and updates ACP Status table.

25. VMS deletes the process and recovers the resources used by the ACP.

2.3 CG/PLNM DATABASE

NOTE

Other nodes on the LCN cannot directly access data in the VAX. Other nodes can, however, read data that has been stored by VAX-resident programs to the CG/PLNM's ACIDP and CRDP data points.

2.3.1 Advanced Control Interface Data Points (ACIDPs)

The Advanced Control Interface Data Point (ACIDP) has the multiple duties of controlling the execution of ACPs in the VAX, of being a message buffer between the VAX and other LCN nodes, and of providing Custom Data Segments for storage of calculated values from the VAX or other LCN nodes (see heading 4.2).

Each ACIDP can be connected to a single ACP in the VAX, or none at all. The ACIDP's Custom Data Segments are normally used to hold values associated with the specific program connected to that ACIDP. However if the ACIDP's Access Key value permits LCN writes, the connected ACP can write to the parameters of any point on the LCN unless the target point's access is limited to Entity Builder only.

2.3.1.1 Impact on Data Transfers

Access to LCN data is affected by the status of a connected ACIDP. No application program can write data to an LCN unless that program is connected (directly or indirectly) to an ACIDP and that ACIDP is in RUN state at the time of the write request. (An ACIDP is placed in RUN state by the CM50S trap handling routine when the program connected to it is legally activated.) Writing to an LCN also requires that the ACIDP be built with an Access Key of READWRITE.

Retrieving data from an LCN does not require an ACIDP connection, but if an ACIDP connection exists (for DDT and Single Point calls) or is explicitly named (for MPL and Point List calls) the data transfer will be blocked at run time unless the ACIDP is in RUN state.

2.3.1.2 Impact on Program Activation

A program that is connected to an ACIDP can be activated only when that ACIDP is in the OFF/DELAY state and is in PERMIT mode. Requests for program activation while the program is running are maintained in the ACIDP, allowing immediate reactivation when the current processing cycle is completed. If an abnormal termination of a program places an ACIDP in Abort state, then all requests to activate that program are ignored until the ACIDP is reset (presumably after the cause of the program failure has been identified and fixed).

The process operator can prevent a program's activation from the LCN by selecting the INH_STAT target on the associated ACIDP's Universal Station Detail Display and setting it to INHIBIT. The same target is used to unblock program activation by setting it to PERMIT. Alternatively, you can construct a custom display that performs this function by a store of INHIBIT/PERMIT to the ACIDP's INH_STAT parameter.

NOTE

Because the INH_STAT parameter is initially set to the "inhibit" state, first-time CG/PLNM scheduling of an ACP cannot begin until after operator action to "permit" operation through the ACIDP's Detail Display or a custom display allowing access to that parameter. See the *CG Parameter Reference Dictionary* for details.

• ACIDP Scheduling

Scheduled activation of a program is requested by the CG/PLNM only if its associated ACIDP shows it to be installed and in Normal (not Restricted or Test) mode, and not inhibited by Process Operator action.

Periodic programs first run at a specified daily start time (STIME) and thereafter run at a specified time interval (RTPERIOD). The STIME value must be less than RTPERIOD. The next scheduled activation time is calculated from the start time and time interval.

Example 1: RTPERIOD = 24:00:00 STIME = 17:00:00
This program runs each day at 17:00:00 hours.

Example 2: RTPERIOD = 08:00:00 STIME = 07:00:00
This program runs each day at the following hours: 07:00:00, 15:00:00 and 23:00:00.

Cyclic programs run at a specified time interval (RTPERIOD). The next scheduled activation time is calculated by adding the specified time interval to the current LCN time whenever the ACIDP is activated (whether by schedule or demand).

Example 3: RTPERIOD = 00:10:00
This program runs every 10 minutes.

The time-interval range for both periodic and cyclic programs is 10 seconds to 24 hours. The subcategories of periodic/demand and cyclic/demand programs also allow for activation by process-operator demand from the Universal Station.

Note that this scheduling is done by the CG/PLNM based on the LCN's clock. Time change at the VAX does not affect scheduling of programs that stay in phase with LCN processing.

• Process Initiated Activation

When a node on the LCN stores a value of 'ON' in the PPS parameter of an ACIDP, the program connected to that ACIDP is immediately activated. Stores to the PPS parameter generally come from an HG (Event Initiated Processing) or an AM (CL/AM program), but

another program running on a VAX can also use this mechanism. This activation method is also used by the CG/PLNM upon receipt of a Message to an ACIDP. Event-activation by PPS is independent of scheduling type or installation mode.

• LCN Operator Activation

An ACP can be activated from a Universal Station through the PROCESS target on its ACIDP's Detail Display if its activation type is demand, cyclic/demand, or periodic/demand. You can also create custom displays that provide for the activation of programs using the OPER_DMD parameter of the ACIDP. See the *Picture Editor Reference Manual* for details.

Operators may also use the PROCESS target to reset an ACIDP from Abort state to Off/Delay.

• Run on Initialization

If an ACIDP is built specifying RUN_INIT = ON and is connected to a program in Normal Mode, it will be activated whenever CM50S is initialized by either a CM50 start up on the VAX or a CG/PLNM initialization/reload on the LCN.

2.3.2 Calculated Results Data Points (CRDPs)

The Calculated Results Data Point (CRDP) role is restricted to that of Custom Data Segment storage for calculated values (see heading 4.2). They generally are used for variables that are not associated with a single program and ACIDP. The use of a CRDP requires the creation a Custom Data Segment defining the parameters that will hold the stored values. The same Custom Data Segment may be assigned to multiple CRDPs.

2.3.3 Resident Data Definition Tables

Two related options provide enhanced performance for data retrieval using a DDT without any changes in the calling program.

Up to 40 data Input DDTs can be memory-resident in each CG/PLNM memory. This eliminates the data-link traffic associated with moving the DDT from the VAX to the CG/PLNM each time it is needed.

You can enable the CG/PLNM to start data collection in parallel with ACP activation. This prefetch of data can be established for any Input DDT stored in CG/PLNM memory by associating it with a specific ACIDP (see heading 6.4).

2.4 OPERATOR INTERFACES TO APPLICATION PROGRAMS

Different types of operator access to operation of an application program are provided at the TDC 3000 Universal Station (process operator) and at the VAX programmer's terminal (computer operator).

2.4.1 Process Operator Interfaces

The process operator can affect application program operation from the Universal Station only if that program is connected to an ACIDP (i.e., the program is an ACP).

The process operator can view ACP status, inhibit ACP operation, demand immediate ACP execution (if permitted for that ACP), view ACIDP and CRDP point.parameter values, and view or change ACIDP and CRDP Custom Data Segment values. The operator's ability to change Custom Data Segment values is controlled by access restrictions established at point-build time.

Some of the ACP-initiated screen messages require operator confirmation.

2.4.2 Computer Operator Interfaces

A user on a VAX terminal can execute CM50S functions through either a menu-driven interface or through command language directives. The menu-driven interface to CM50S functions is described in Section 3, and the use of DCL level commands to invoke CM50S functions is described in Section 8.

The menu-driven interface supplies specialized interactive displays that enable the computer operator to install or uninstall an ACP, change its operating mode, and start and stop individual ACP execution. He also can view ACP status by examination of ACIDP parameters in the CG/PLNM. Other VAX terminal interactive displays affecting application programs are the DDT Operations display, the Task Scheduler display, and the Makeinc Utility display.

2.5 CM50S PROGRAMMATIC INTERFACE

CM50S functions can be invoked by function calls from within a user-written application program. (An exception is the generation and editing of source code for a DDT, which can be done using standard text file handling logic.) The function calls can be made from Pascal, FORTRAN, or "C" programs, providing all arguments are passed by reference.

The calling program does not need to be an ACP, but some functions (particularly activating an ACP) require that the user have appropriate VMS privileges. All users of the CM50S programmatic interface must be registered with the SYSLCK privilege.

The programmatic interface is divided into two sections, one for LCN data transfers (all application programs), and one for DDT and ACP management functions.

2.5.1 Programmatic Interface Include Files

The Programmatic Interface is supported by include files for user application programs that define data structures, constants, and subroutines required to make the Programmatic Interface calls. These include files are language specific. The Pascal definitions establish all arrays at there maximum allowable size; if smaller arrays are used as actual arguments, they should be passed explicitly by reference to avoid conflicts with Pascal's strong data typing restrictions. The C function definitions include prototyping of the arguments.

2.5.1.1 Data Transfer Routines

One of these files will normally be included in each application program:

`CM50$LIB:CM50_INCLUDE.PAS`	Pascal types and definitions for the standard functions.
`CM50$LIB:CM50_DDT_INCLUDE.H`	C definitions for the standard functions.
`CM50$LIB:CM50_DDT_INCLUDE.FOR`	FORTRAN definitions for the standard functions.

2.5.1.2 DDT and ACP Management

DDT and ACP management functions use some shared data structures, which are defined in the CM50_FLAGS_INCLUDE files. Therefore, this file should be included in any program that calls either DDT or ACP functions before the include file defining those specific functions.

`CM50$LIB:CM50_FLAGS_INCLUDE.PAS`	Pascal definitions for the shared data structures in the programmatic interface calls.
`CM50$LIB:CM50_DDT_INCLUDE.PAS`	Pascal types and definitions for all programmatic DDT operations.
`CM50$LIB:CM50_ACP_INCLUDE.PAS`	Pascal types and definitions for all programmatic ACP management operations.

CM50$LIB:CM50_FLAGS_INCLUDE.H	C definitions for the shared data structures in the programmatic interface calls.
CM50$LIB:CM50_DDT_INCLUDE.H	C definitions for all programmatic DDT operations.
CM50$LIB:CM50_ACP_INCLUDE.H	C definitions for all programmatic ACP management operations.
CM50$LIB:CM50_FLAGS_INCLUDE.FOR	FORTRAN definitions for the shared data structures in the programmatic interface calls.
CM50$LIB:CM50_DDT_INCLUDE.FOR	FORTRAN definitions for all programmatic DDT operations.
CM50$LIB:CM50_ACP_INCLUDE.FOR	FORTRAN definitions for all programmatic ACP management operations.

2.5.1.3 LCN File Transfer Routines

The LCN File Transfer functions share some data declaration with the DDT and ACP management functions. These common declarations are defined in the CM50_FLAGS_INCLUDE files. Therefore, this file should be included before the CM50_FTF_INCLUDE file in any program that calls a file transfer program.

CM50$LIB:CM50_FLAGS_INCLUDE.PAS	Pascal definitions for the shared data structures in the programmatic calls.
CM50$LIB:CM50_FTF_INCLUDE.PAS	Pascal declarations specific to LCN file transfer functions.
CM50$LIB:CM50_FLAGS_INCLUDE.H	C definitions for the shared data structures in the programmatic calls.
CM50$LIB:CM50_FTF_INCLUDE.H	C declarations specific to LCN file transfer functions.
CM50$LIB:CM50_FLAGS_INCLUDE.FOR	FORTRAN definitions for the shared data structures in the programmatic calls.
CM50$LIB:CM50_FTF_INCLUDE.FOR	FORTRAN declarations specific to LCN file transfer functions.

2.5.2 Programmatic Interface Flag Parameter

A 32-bit parameter called FLAGS (in FORTRAN declare as INTEGER*4) is included in every programmatic DDT and ACP management function to control some of the handling options. Some of the flags apply to only the DDT calls, some to only the ACP calls, and some can be used by both. All user-visible flags are described below.

- CM50$M_HANDLER—(Bit 0) Indicates that the user has provided a custom exception handler. The default is OFF.
- CM50$M_MSGON—(Bit 1) Prints all diagnostic messages to SYS$OUTPUT. The default is OFF.
- CM50$M_CGRES—(Bit 5) Installs the DDT as CG/PLNM resident. The default is OFF.
- CM50$M_REBUILD_DDT—(Bit 6) Rebuilds an existing DDT. The default is OFF.
- CM50$M_NO_SOURCE_DEBUG—(Bit 7) Produces no error file during DDT build. The default is OFF.
- CM50$M_DMP_DDT_ERRORS—(Bit 8) After building the DDT, sends the error file produced by the DDT build to SYS$OUTPUT. If this flag is set, then the CM50$M_NO_SOURCE_DEBUG flag must be OFF.
- CM50$M_ACIDP_ACTIVATE—(Bit 9) Reserved for internal CM50S use.
- CM50$M_WRITE_VT—(Bit 10) Creates the .VT file with write privilege.

All of the flags described above, represent bit masks that can be added together to enable any combination of the flags. These flag values also can be used to see if a particular flag is set. A Pascal example is shown below.

```
%include 'CM50$LIB:CM50_FLAGS_INCLUDE.PAS'
%include 'CM50$LIB:CM50_DDT_INCLUDE.PAS'

var
   DDT_Name        :  DDT_NAME_TYPE;
   Summary         :  DDT_SUMMARY_REC;
   Flags           :  CM50_FLAG_TYPE;
   Return_Status   :  FUNC_RET_STAT;

begin
   DDT_Name := 'test     ';
   flags    := 0;
   .
   .
   flags := CM50$M_HANDLER + CM50$M_MSGON;
   Return_Status := DDT_SUMMARY(DDT_Name, Summary, Flags)
end;
```

2.5.3 Linking Application Programs to CM50S

User application programs must be linked with the programmatic interface routines in order to invoke those functions.The following link command links the TEST application program to all the programmatic interface routines:

```
1 $ LINK /EXEC=USER_DIRECTORY:  USER_DIRECTORY:TEST -
2                            CM50$LIB:CM50_CONTROL.OBJECT,-
3                            SYS$INPUT/OPT
4                            CM50$EXE:CM50_SHARE.EXE/SHARE
5                            CM50$EXE:CM50_FTF_SHARE.EXE/SHARE
```

Line 2 directly links the CM50S trap handling routines to the executable program.
Line 4 maps the program to the CM50S global common and is required for every program that uses any of the programmatic routines.
Line 5 maps the program to the File Transfer global section and is required for programs that issue calls to any of the LCN file transfer functions.

2.6 IMPLEMENTATION STEPS

The exact sequence varies depending on specific circumstances, but all the following steps are necessary to complete the installation of an application program. Number references included in the following headings point to later sections in this manual where more detailed information is found.

NOTE

You should consider initiating a checkpoint of the CG database any time that it is modified (by the addition or deletion of points, by ACP connect or disconnect, by creating a CG/PLNM-resident DDT, etc). If there is no checkpoint, either demand or automatic, the CG database reverts to the previous contents at its next restart.

2.6.1 System Start Up

Follow the instructions in the CM50S Release Notes to configure the CG/PLNM(s), then see Appendix B in this manual for instructions on adding CM50S software to the VAX/VMS computer system, and the starting of VAX to CG/PLNM communications.

2.6.2 Prepare CG/PLNM-Resident Data Points (Section 4)

The ACIDP and CRDP point types are built at a Universal Station, using the Engineering Personality (preceded by use of the CL compiler to define any Custom Data Segments associated with them). Installed ACIDPs and CRDPs can be displayed at a Universal Station (Point Detail Display) and at the VAX (using the CG Database displays).

2.6.3 Prepare Data Definition Tables (Section 6)

Data Definition Table source files are built or modified at the VAX, using the DDT Operations and Edit screens. Any referenced points must be built and be accessible through the data owner before the table build can be completed. ACP references to nonexistent or incomplete DDTs are rejected by the interface routines.

2.6.4 Compile, Link, and Install Programs (Headings 5.1 - 5.6)

The interface routines are designed to support programs written in either FORTRAN, Pascal, or "C." Each application program is separately compiled and linked; ACPs must also be installed in the CM50S system.. Compilation is done through standard VMS software; a special CM50S .COM file is used for all linking of CM50S programs; installation is done through the CM50S ACP Operation screen.

2.6.5 Test and Modify Programs and DDTs (Heading 5.7)

Program testing is assisted by the two special installation modes, "Test" and "Restricted." Data Definition Tables can be viewed by use of the DDT Detail Description and List displays. Certain ACIDP parameters can be manipulated from the Universal Station Point Detail Display.

ACCESS AND SECURITY
Section 3

This section describes the procedures necessary to access CM50S from the host computer and the security measures that are supported to limit access for selected uses.

CM50S operates as a layered product on top of VMS. It uses the standard VMS mechanisms for user access and security. The section describes the specific requirements for CM50S. For more detailed information on the security mechanisms see Guide to VMS System Security (DEC order number AA-LA40B-TE). In general, each site should implement the minimal set of measures that meet the actual security needs of its environment, because use of advanced security options 1) adds system overhead (degrading overall performance), 2) requires additional staff time to administer, and 3) makes the system more difficult to use.

3.1 USER ACCESS TO CM50S

3.1.1 User Registration Requirements

In order to access CM50S from a VAX terminal, a person must be a registered user on the VAX system and must be granted a VMS Rights Identifier of "CM50S_SYS." In order to activate an ACP interactively, the user must also have the VMS "TMPMBX" privilege. Any program (process or image) that makes calls to CM50S must have the VMS "SYSLCK" privilege. (If a user is going to interact with CM50S only through the menus and DCL interface, then he does not need to be registered with the SYSLCK privilege because that privilege is granted to installed CM50S images.) In addition, it is recommended that each user be registered with a distinct home directory, to minimize problems of maintaining work files.

The standard installation procedure for CM50S protects all of the CM50S files with an ACL (VMS Access Control List) that limits access to users with the "CM50S_SYS rights identifier (or the VMS "BYPASS" privilege).

Note: Earlier releases of CM50S required users to have the VMS "WORLD" and "DETACH" privileges. These privileges do not have to be granted to users of the current CM50S release.

3.1.2 Log In Procedures

Logging on to use CM50S requires only the standard VAX log in procedures of typing in a user name and then a password. Appropriate assignment and protection of passwords must be considered the foundation for all terminal access security.

Use of CM50S is made simpler if the user's LOGIN.COM file automatically points him to components of the CM50S software by including the following sequence of commands:

```
$! check to see if cm50 is running:
$ cm50_ok = f$search("CM50$LIB:CM50_ERROR_MSG.EXE;0")
$ if cm50_ok .eqs. "" then goto NO_CM50S
$ set message CM50$LIB:CM50_ERROR_MSG
$ set command CM50$LIB:ACP_COMMAND
$ set command CM50$LIB:DDT_COMMAND
$ set command CM50$LIB:FTF_COMMAND
$ goto CM50_DONE
$ NO_CM50S:
$! put out an appropriate message
$ CM50_DONE:
$! continue with other log in commands
```

To have a user start off with the CM50S main menu (described below), include the following command in his LOGIN.COM file: $ @CM50$LIB:CM50

3.1.3 Menu-Driven Operations

All CM50S interactive functions can be accessed through the CM50S Main Menu (invoked by the DCL command line: @CM50$LIB:CM50).

```
CM50S R400                      CM50S MAIN MENU                  11 APR 91 12:44

AP -- ACP Operations                        UG -- User Guide Index

DT -- DDT Operations                        VS -- Value Status Index

CG -- CG Database Displays                  RS -- Return Status Index

FT -- File Transfer with LCN

SC -- Scheduler

MI -- MakeInc Utility




QT -- Quit (Log Off)                        EX -- Exit (to DCL)

GPF4/EXIT     PF2/HELP       Choice:              PF4/PRIOR    G-Enter/MAIN MENU
```

Any item on this menu can be invoked by either using the arrow keys to highlight the desired entry and then pressing <return>, or by typing the item's two-character entry code and then pressing <return>. Note that the choice field on the screen allows multiple entries (separated by spaces), so the experienced user can skip through additional layers of menus.

For example: an entry of SC CT <return> would jump directly to the Command Table Maintenance function (entry CT on the Task Scheduler submenu).

The CM50S Main Menu provides access to five CM50S function categories and three indexes. The functions are:

(AP) The ACP Operations displays (see heading 5.2) can also be invoked by the command line: RUN CM50$EXE:ACPOPER. These displays allow you to:

- Install or remove ACPs in the VAX
- Connect or disconnect ACPs to ACIDPs in the CG
- Activate or deactivate/terminate ACPs
- Change an installed ACP's status (Normal, Restricted, Test)
- Examine the ACP Status Table
- Examine the ACP Installer activity file

(DT) The DDT Operations displays (see heading 6.2) can also be invoked by the command line: RUN CM50$EXE:DDTOPER. These displays allow you to:

- Create and modify DDT source files
- Build or rebuild DDTs from the DDT source files
- Examine contents of DDTs

(CG) CG Database displays (see heading 4.5) can also be invoked by the command line: RUN CM50$EXE:CGDSP. These displays allow you to:

- Display status of the configured VAX-CG data links
- Display status information on a selected ACIDP
- Display lists of all ACIDPs and CRDPs installed in the CG
- Display a list of all CG-resident DDTs

(FT) The File Transfer facility (see heading 7.3) is used to transfer files between the VAX and the LCN. This facility invokes the LCN file utilities and does not support multiple concurrent users. The available options include:

— Read an LCN file into the VAX
— Write a file from the VAX into an LCN History Module
— List the file and directory names from an LCN History Module
— Copy, Move, Rename, and Delete files on an LCN History Module
— Create and Delete a directory on an LCN History Module
— Abort a file transfer operation that is in progress

(SC) The Task Scheduler menu (see heading 7.1) is used to

- Start the Task Scheduler
- Stop the Task Scheduler
- Modify the Scheduler's Command Table
- Modify the Scheduler's Configuration value

(MI) The Makeinc utility (see heading 7.2) can also be invoked by the command line: RUN CM50$EXE:MAKEINC. It allows you to create "include" files that are used with certain types of data get/store user interface routines.

The three on-line indexes are available to look up references in the *CM50S User Manual,* and to translate LCN Value Status and CM50S Return Status codes. Each of these menu options prompts for appropriate search criteria (words or character strings). To search for multiple words with a single entry, separate the words by commas; then only occurrences that contain all of the listed words are displayed. Search strings that contain spaces or other special characters must be enclosed in double quotes (" "). The three on-line indexes are:

(UG) The User Manual index locates sections of the *CM50S User Manual* based on keywords. When you choose this option, CM50S prompts you to enter a word to search for. Reply to this prompt by typing a word (or any string of characters) and press <return>. All entries in the index to the user manual that contain the specified character string are displayed.

(VS) The Value Status index translates LCN codes that are returned with each value transferred to (or from) the LCN. This index includes the File Manager codes (prefaced by FILE-) and Utility Manager codes (prefaced by UTIL-) that are returned as secondary error status from the File Transfer routines (in addition to the LCN Data Access codes). Response to the prompt for this option is the three-digit numeric value of the LCN code. Be sure to include any leading zeros (e.g., 006, not just 6).

(RS) The Return Status index translates the return status codes for CM50S functions. Response to the prompt for this option is either mnemonic error condition name or the decimal integer value of the status code (e.g., CM50_ACP_EXEC or 215001786). Note that the index does not contain the hexadecimal representation of these codes.

3.1.3.1 Display Screen Format

All CM50S display screens share the following format characteristics:

- The top line of every screen identifies the application and includes date and time. Note that the time shown indicates when the display data was last refreshed. The clock does not update for static displays.
- Active function keys are shown on lines 22 and 23 of the display.
- Interactive messages and prompts appear on line 24 of the display.
- All fields into which data can be entered appear underlined on the form and the current field for data entry is highlighted by reverse video.

3.1.3.2 Screen Data Entry

Whenever a CM50S display appears on your screen, the cursor will be positioned at the first data entry point. You then enter/modify the required data through the keyboard, then press the Enter key on the keypad at the right-hand side of the keyboard. Depending on the display, you may need to follow this by use of one of the function keys:

- Left/Right Arrows—move the cursor one character position
- Backspace—delete prior character
- Up Arrow—move cursor to prior field
- Down Arrow, Tab, or Return—move cursor to next field

3.1.3.3 Use of Function Key Menus

A function key menu appears at the bottom of every CM50S screen. Each function key menu shows the names of the valid function keys for that screen, along with an abbreviated description of what action the function key performs. These functions are invoked by using the keypad on the right-hand side of the keyboard. Function selections prefixed by KP indicate a numbered key on the keypad (KP2 = keypad key 2). Function selections prefixed by PFn indicate one of the keypad keys PF1 through PF4. Function selections prefixed by G indicate that the Gold key (PF1) must be pressed before pressing the key following (G. = press PF1, then the "." keypad key). Real values can be entered as integers, fixed decimal point, or using exponential notation.

Screen menu entries can be selected either by typing the entry code or number into a selection field or by moving the cursor to the desired entry (it will be highlighted in reverse video) and then pressing Enter. Function keys are labeled at the bottom of the screen. For displays with more function capabilities than can be shown at one time, the PF3/MORE key brings up an alternate function key menu for that display. Note that the full set of function keys for a display is always active, independent of which partial set is shown. Two function keys that appear on the Main Menu have a consistent meaning across all CM50S displays:

PF4/QUIT—Cancels data entry and backs up one level in the display tree. (In this instance, exits the CM50S Interactive User Interface and returns to DCL or logs off, depending on the user registration option.) In some instances, you will be asked to verify that you wish to quit without saving changes.

PF2/HELP—Displays a short help message at the bottom of the screen. Press PF2 again for a display describing the entire form. Pressing <PF4> from this screen returns to the originating display.

NOTE

Because of space limitations, the PF2 / HELP function may not be shown on every screen; however, the PF2 key provides Help functions for all CM50S screens.

3.1.4 DCL Commands and Error Messages

CM50S includes support utilities that allow the execution of ACP and DDT maintenance functions and translation of the CM50S error codes into text messages. To enable the DCL command interface, the user (or the LOGIN.com file) must first map the CM50S commands to the command processor by
"SET COMMAND CM50$LIB:ACP_COMMAND" and
"SET COMMAND CM50$LIB:DDT_COMMAND" and
"SET COMMAND CM50$LIB:FTF_COMMAND" for the ACP, DDT, and File_Transfer functions respectively. The custom commands (described in Section 8) may then be invoked interactively or through command procedures exactly like standard DCL commands.

"SET MESSAGE CM50$LIB:CM50_ERROR_MSG" maps the CM50S error messages to the system message handler. This makes translation of the status codes accessible through the F$MESSAGE lexical function. For example, the commands:

```
a = F$MESSAGE (215000036)
SHOW SYMBOL a
```

would display the text error message "Unable to access LCN -- datalink failure."

3.2 STARTING UP CM50S

No user can access CM50S unless its shared images are installed and the base communications processes are running. These tasks are accomplished by the protected CM50_STARTUP.com procedure. As normally installed, this procedure may only be run from a privileged user account named CM50S_MGR (or SYSTEM).

3.2.1 Using the CM50S_MGR Account

The CM50S_MGR account is established as part of the normal CM50S installation. This account is the owner of all the shared files created by CM50S. Since it is a privileged account, the system administrator should carefully control its password.

NOTE

The CM50S_MGR must be granted the "CM50S_SYS" rights identifier and all of the VMS system privileges (including SETPRV) active.

Before CM50S can be accessed from the user level, certain tasks must be performed to establish the databases and synchronize with the CG. In order to ensure that the detached processes have all the appropriate privileges and quotas and create files with the standard protections, the CM50S startup procedure must be run under the UIC of an account named CM50S_MGR (or SYSTEM).

From the DCL prompt, the CM50S_MGR starts the CM50S software with the command:

```
$@CM50$:[R040]CM50_STARTUP.COM
```

This procedure assumes the existence of a logical volume name "CM50$" pointing the the root directory for the CM50S software. The "[R040]" specifies the release subdirectory. A release subdirectory carries the name of the version number (e.g., R040 for Release 4.0).

If for any reason the CM50S startup procedure is terminated prematurely, it will be necessary to execute the CM50_STOP.COM procedure (or to reboot the VAX) to release memory-resident modules before CM50S can be restarted successfully.

3.2.2 Submitting a Start Up Request

In most sites it is desirable to start CM50S as part of an automatic procedure run by the system manager without having to log on to the VAX system specifically as the CM50S_MGR. This can be accomplished by a privileged user submitting the request to be run under the UIC of the CM50S_MGR. Submitting a start up request requires two commands:

1. Define a system-wide logical device name CM50$ pointing to the CM50S root directory.

```
$ DEFINE/SYSTEM/EXEC/NOLOG -
        /TRANSLATION=(TERMINAL,CONCEALED) -
         CM50$ DUA0:[CM50.]
```

2. Invoked the CM50$[RO40]CM50_STARTUP.COM command procedure either directly within the REBOOT command procedure or by submitting it to a batch queue. This startup can only be done from the CM50S_MGR or SYSTEM account. In either case, the CM50S processes will be run under the CM50S_MGR UIC.

   The "[R040]" specifies the release subdirectory. A release subdirectory carries the name of the version number (e.g., R040 for release 4.0).

Usually it is desirable to have CM50S activated automatically when VMS boots. This can be achieved by placing the two defined commands in the system-wide startup command procedure.

3.3 SHUTTING DOWN CM50S

NOTE

This function must be performed from the CM50S_MGR account with full system privileges (including SETPRV) active.

Because critical database changes are checkpointed as they occur, it is not necessary to shut down CM50S before shutting down VMS; however, if the need arises to shut down CM50S without rebooting VMS, the command procedure CM50_STOP.COM can be invoked from the CM50S_MGR account. This procedure resides in the release subdirectory.

```
$@CM50$VER:CM50_STOP
```

This operation deactivates all detached processes, removes all installed images, and effectively purges CM50S from the operating system. It is necessary to invoke this procedure when a new release of CM50S is loaded, before its activation.

3.3.1 User Notification

All users that are mapped to the CM50S shareable images will be given two warning messages when the CM50S product is being shutdown. These messages will be sent to all users whose mode is determined to be "interactive." For example, a user with the CM50S Menu Screen displayed would receive a warning; a user running an interactive ACP with a login terminal name (JPI$_TERMINAL) would also receive a message.

3.3.2 User Specified Delay

The option of a minimum delay period between Notification Messages has been provided. The default minimum delay time is 00:00:00. The delay period is in effect after all the interactive users have received the first warning message. For example it may require 30 seconds to notify all the interactive users, followed by a 1 minute user-specified delay period, then another 20 seconds to issue the second warning message to users still connected to CM50S. This 00:01:50 second delay period is followed by a second 1 minute user-specified delay period. Thus, the elapsed time before the first user is force-terminated by CM50S is 00:02:50

3.3.3 User Friendly Termination

Previous releases of CM50S performed a "STOP/ID=xxxxxxxx" termination on all processes found mapped to CM50S sharable images. This modified procedure will now issue a "forced exit" command and allow the user the option of executing a user-written procedure. This method of image shutdown is preferable to an arbitrary stop/id=xxxxxxxx exit. Users should not create extensive time consuming exit handlers because the next time the delay period expires, the process will be terminated using the stop/id=xxxxxxxx scenario.

3.3.4 Command

The CM50S stop procedure has not changed; that is, entering the command "@CM50_STOP.COM" will still shutdown the system. Two additional arguments are supported. Typing a question mark at the end of the command will provide on-line help of the stop procedure command line format. The first optional argument is a [CONFIRM Y/N]; the stop procedure will display information on processes mapped to CM50S. The default is [N]. The second argument permitted is the User-Specified Delay. The period must be in the format HH:MM:SS. This is a minimum delay time period that is added to the normal message notification period.

NOTE

If the delay period is not defaulted and the confirm is defaulted, the confirm default is [Y].

EXAMPLE:	COMMAND	CONFIRM	DELAY	
	@CM50_STOP	**[Y/N]**	**[HH:MM:SS]**	
	@CM50_STOP	**[N]**	**00:00:00**	**(*DEFAULT)**

3.4 LIMITING ACCESS TO FUNCTIONS

3.4.1 Restrictive Rights

At sites where the security requirements distinguish different levels of CM50S access for different users, the system administrator may assign Restrictive Rights identifiers to selected CM50S user accounts to prevent them from performing specific functions. By default, any user with the CM50S_SYS rights identifier may perform any CM50S function.

To implement this additional layer of security, the system administrator must create a VMS Rights Identifier for each of the restrictions to be implemented and grant those identifiers to the user accounts which are **not** to be permitted to perform the specific functions listed below:

Rights Identifier	Prohibited functions
NO_ACP_INSTALL	Install or Uninstall an ACP Install or Remove DDT from CG Connect or Disconnect ACP or DDT with ACIDPs Change ACP installation mode Change DDT prefetch triggers
NO_ACP_START	Activate an ACP at a VAX terminal
NO_ACP_STOP	Deactivate an ACP
NO_DDT_CREATE	Build, Edit or Delete a DDT
NO_LCNFILE_READ	Read LCN Files (or use DataOut)
NO_LCNFILE_WRITE	Modify files on the LCN (Write, Copy, Move, Rename, Delete, Create directory, or Abort transfer)

3.4.2 Access Control Lists

Access to specific ACPs and DDTs can be controlled by applying VMS ACLs (access control lists) to the appropriate files. Care must be exercised in assigning ACLs since, if they are overly restrictive, they can prevent the remote execution of an ACP triggered from the LCN.. If the ACPs are installed to run under a specific UIC, then that UIC must have access to both the ACP and DDT files; otherwise, the default user "CM50S_MGR" must have access to all the files needed by the ACP.

The DDT files (except the user's source) are maintained in the CM50$DDT directory with the file name equal to the DDT name and different extensions for different functions. If the DDT was built with a values table specified, then the file with extension .VT must be available to the ACP or Data Access Program with Write access. Read access on the other DDT files is sufficient for all operations (except building/rebuilding or deleting the DDT).

3.5 ACP SECURITY

Installing a program as an ACP allows different users to activate it remotely from the LCN. Thus, there are several security mechanisms that are designed specifically for ACPs.

3.5.1 ACIDPs

ACIDPs provide security within the LCN to limit the execution of remote programs. In order for any program in the VAX to be activated from the LCN, or to store any data to the LCN, that program must be installed as an ACP and connected to an ACIDP. ACIDP configuration on the LCN determines whether the program can be activated from a Universal Station and/or from the LCN scheduler. Note that any ACIDP can be triggered by CL code, so it is not safe to assume that an ACIDP that is configured for only PERIODIC triggers is only activated by the scheduler. The details of ACIDP security are described in section 4.

3.5.2 Limiting Triggers

An ACP can limit its own functioning based on the way it was activated. It does this by issuing a call to the GETSTS procedure (FORTRAN section 11.1.2, Pascal section 15.1.2, or C language section 19.1.2) to determine which trigger(s) caused the current activation; the ACP then takes appropriate action.

For example: An ACP is designed to be executed only when requested by an operator on a Universal Station. Before performing any data transfers, it would call GETSTS and test the returned **demand** argument. That argument will be true only if the ACIDP was triggered by an operator. If **demand** were false, the ACP would terminate, preventing its functions from being initiated by a VAX user or the LCN scheduler.

3.5.3 Running Under Specified Accounts

By default, a remotely activated ACP runs under the CM50S_MGR account, using the file protections assigned to that account. If it is desired, a different user account can be specified when the ACP is installed. The account used for a remote ACP activation will determine which group flags it can reference as well as the file protections. Note: A user must have VMS privileges (Group or World) to install an ACP under a user account other than the default CM50S_MGR or his own account name.

3.5.4 Privileges and Quotas

By default, a remotely activated ACP will have only the SYSLCK VMS privilege and the default user quotas. In order to activate an ACP interactively, the user must have the VMS "TMPMBX" privilege (which allows the creation of a termination mailbox to receive notice of the ACP's completion). If an ACP needs special privileges or quotas, they can be assigned at the time the ACP is installed, with the limitation that no user can install an ACP with privileges that are not assigned to his own account.

3.5.5 ACP Installation Log

The ACP Installation Log maintains a history of actions which affect the installation status of any ACP. The actions that are recorded are:

- ACP Installation
- ACP De-installation
- ACP Connect to an ACIDP
- ACP Disconnection from an ACIDP
- Change Installation Mode of an ACP
- Interactive ACP Activation
- ACP Deactivation
- DDT Connection to an ACP for Prefetch
- DDT Disconnection from an ACP
- Change of DDT prefetch triggers

NOTE

Changes in the ACP code are not logged until the ACP is reinstalled. To maintain this information in the log, the VMS file version number should be specified as part of the execution path whenever ACPs are installed.

The ACP Installation log is maintained as a direct access text file (CM50$CONTROL:ACPI_REC.DAT) with an associated set of pointers (CM50$CONTROL:ACPI_PTR.DAT). The system administrator can configure the number of actions to be maintained in the log by running the CM50$SUPPORT:CM50_ACPLOG_CONFIG utility from the CM50S_MGR account. This utility will prompt for the desired size of the log and create a new log file of the appropriate size. ACP activity logging can be suppressed, causing a trivial improvement in performance, by specifying a log size of zero. By default, the ACP Installation log will contain the most recent 1000 entries.

CM50S does no archiving of prior versions of the Installation log, and will overwrite old entries with new data once the log has been filled. Archiving of the log files (if desired) is the responsibility of the local system administrator.

3.6 SENDING MODIFIED FILES TO THE LCN

The File Transfer software restricts the types of file modifications permitted at the History Module. You cannot read a binary file from the LCN, modify it at the VAX, and then write it back to the LCN, since the modification could cause the node to crash. A file that has been edited/modified on the VAX can only be written to an LCN if its extension identifies it as an ASCII file.

Every file to be transferred to the LCN must have an LCN attributes file. This is created at the time of the original transfer. No software tools are provided to allow the origination of a file of either type (ASCII or binary) and its LCN attributes at the VAX. If the file is of type binary the attributes are checked against the original LCN file attributes preserved when the file was first transferred. If they are different, a transfer error will occur.

3.6.1 ASCII File Extensions

The following file extensions have been identified as default ASCII file types. They are specified in the CM50$CONTROL:FTF_CONFIG.DAT file that is installed with CM50S. Files with any of these extensions can be modified on the VAX and returned to the same or a different LCN running the same release of TDC 3000. Note that changes in the LCN file attributes between releases of the operating system can prevent the transfer of a file back to the LCN.

FILE SUFFIX	LCN FILE TYPE
BU	Backup Text
CL	CL Source
EB	Exception Build Source
EC	Execute Command
EF	Error File (DEB)
EL	Edited List, Entity Names
ER	Error Report Buffer File
JL	Logic Block Listing
JS	Logic Block Source
LE	CL Error Listing
LS	CL Listing
SL	Successful Entity List
UL	Unsuccessful Entity List
X	User Text
Y	User Text
Z	User Text

3.6.2 Customizing File Extension Control

Additional files can be modified and transferred to the LCN if their extensions are placed in the CM50$CONTROL:FTF_CONFIG.DAT file. The standard installation restricts updating of this file to the CM50S_MGR (or other fully privileged VMS SYSTEM account). These changes will become effective dynamically (that is, there is no need to shutdown and restart CM50S for the new list of extensions to become effective).

This one record file contains an array of up to 40 file extensions that define an "ASCII file" type. The File Transfer software checks this file to verify that the file to be written to the LCN is of type ASCII. If the file extension is not in the FTF_CONFIG.DAT then it is considered to be a binary file which may only be written to the LCN if it has not been modified.

The "ASCII" extensions are written as a single string of characters. Each pair of characters represents an extension that will be treated as an ASCII file. One-character extensions are followed by a space; otherwise, there are no delimiters between extensions. The characteristics of the file are:

File Organization:	sequential
Record Format:	variable
Maximum Record Size:	255
Record Length:	80 bytes max
Record Attributes:	carriage-return

CG/PLNM POINT PREPARATION
Section 4

This section summarizes the requirements for preparation of specialized CG/PLNM data points that regulate the execution of control programs and hold results of control calculations for exchange with other LCN nodes.

4.1 CG/PLNM POINT BUILDING OVERVIEW

As explained in Section 2, there are two types of data points that reside in the CG/PLNM, the Advanced Control Interface Data Point (ACIDP) and the Calculated Results Data Point (CRDP). Both point types can be used to store calculated results that are to be exchanged between the CM50S and other LCN nodes. The ACIDP also contains the parameters that control the scheduling and execution of an associated ACP in the VAX. Both point types are built at an Operator Station that is running in the Engineering Personality.

Each CRDP and those ACIDPs that include data storage reference Custom Data Segments (CDS) that define the special parameters to be added to that point; therefore, Custom Data Segments must be prepared in advance of ACIDP/CRDP point building.

4.2 CUSTOM DATA SEGMENT CONSTRUCTION

NOTE

The following information, on preparation of Custom Data Segments, is intended only as an introduction. Please consult the *Control Language/AM Reference* Manual for details of CL program preparation and the *System Control Functions* manual for additional information on Custom Data Segments.

Custom Data Segments allow you to define new (nonstandard) parameters and add them to data points. Once you define new parameters and add them to data points, they can be accessed in the same manner as standard parameters. Up-to-10 Custom Data Segments can be associated with any ACIDP or CRDP.

Custom Data Segments are constructed as Control Language/AM (CL/AM) Packages, each consisting of a single CDS. These CL/AM "packages" are compiled, then stored on the History Module or on Floppy Disk for use when the individual data points are built. The Data Entity Builder (DEB) is used to add instances of a CDS to one or more data points.

Each CDS consists of a Heading, plus one or more parameters.

4.2.1 Custom Data Segment Heading

The Custom Data Segment Heading consists of the word CUSTOM followed by three optional attribute assignments, which change the default values for Class, Access, and Build Visible for this CDS. Either the standard or heading-specified default values are overridden by any individual parameter attribute assignments. Always use the default value for Class when preparing a CDS to be used with an ACIDP or CRDP.

4.2.2 Custom Data Segment Parameters

Each CDS parameter has a heading that begins with the word PARAMETER followed by an up-to-8-character name, an optional data-type specifier, and an optional character string to be displayed by the DEB. This is followed by a set of optional attribute assignments.

Data type can be Number, Time, Logical, Enumeration, String, or Data Point Identifier (or single-dimension arrays of any of these). The default data type is Number.

The parameter attributes are

ACCESS—The Access Attribute defines write access restrictions for the parameter. Read access is never restricted. The access levels are View Only, Operator, Supervisor, Engineer, Program, and Entity Builder. The standard default access level is Engineer. For additional information on parameter access-level significance, see the *System Control Functions* manual.

BLD_VISIBLE (or NOT BLD_VISIBLE)—Determines whether or not a preset parameter value can be changed at point-build time. The standard default value is BLD_VISIBLE.

VALUE—The data type of the constant expression must match the parameter's assigned (or default) type. If no VALUE is specified and NOT BLD_VISIBLE is specified, a default value is assigned. The default values vary by data type as specified in the *Control Language Reference Manual*.

EU—The Engineering Units attribute is a character string that is displayed with other point.parameter information. The default is blanks.

CLASS—For an ACIDP or CRDP CDS, always use the standard default value of General.

4.2.3 Custom Data Segment Example

```
CUSTOM

PARAMETER swdbd1:NUMBER "switch deadband value"
ACCESS engineer
EU "psi"
VALUE 0.5
BLD_VISIBLE

PARAMETER swdbd2
EU "psi"
VALUE 0.5

END CUSTOM
```

Notice that the two parameters generated by this example are identical in all but their names. There is no name associated with the CDS because it is identified by the name of the file into which it is compiled (only one CDS per file).

4.2.4 Custom Data Segment Compilation Recommendation

The CL compiler maintains a library file that includes the names of all nonstandard parameter names used in every CDS ever compiled in your system. This file allows for 1000 names, which is normally more than adequate because any of these parameters can be arrays of values and a particular name is entered only once no matter how many times it is used by multiple Custom Data Segments.

Once a name is entered into the library file, however, there is no convenient way to delete the name. Because it is not desirable to clutter the file with parameter names that were accidentally mistyped, the compiler does not update the library file unless the compiler directive -UL (Update Library) is invoked. You should obey the following sequence when compiling a CDS:

1. First, compile **without** the -UL directive to ensure that the CDS parameters are free of errors. Every new parameter is followed by an error indicating that the -UL option should be used. If any **other** errors appear, they should be corrected.

2. Recompile **with** the -UL directive to update the system library file with the CDS parameter names. There should not be any errors.

It is a good idea on subsequent recompilations to compile without the -UL directive unless a new parameter name is purposely being added to the CDS. This guards against erroneous additions that might occur if a parameter name is accidentally mistyped while editing the file.

You can see all the parameters that have been defined by using File Manager Utilities to print the system library file &ASY>PARAMETER.SP. The file &ASY>SEGMENTS.SP can be printed to see all CDS file names that have been used. The -UL directive also controls whether CDS file names are entered into the library.

Compiling a CDS does not set aside storage for the parameter values; it simply defines the parameters to the system. The next required step is to build a point that uses the CDS parameters. Once a data point is built, the parameters of the CDS are part of the data point and are undifferentiated from other parameters of the data point.

At this point, you should back up the .SE and .SP files on &ASY, and should also checkpoint the CG portion of the CG/PLNM database.

4.3 ACIDP/CRDP POINT BUILDING

CG point data can be recorded on *Computer Gateway Forms* in preparation for the actual point-configuration process. Explanation of the entries is found in the *Computer Gateway Parameter Reference Dictionary* and the point-entry process is described in the *Data Entity Builder Manual.*

A brief outline of ACIDP/CRDP building follows:

1. From the Engineering Personality Main Menu, select the `COMPUTING MODULE` target in the Point Building column. This calls up the `CM BUILD AND CONFIGURATION` menu.

2. Select the target appropriate to the point type to be built.

3. Enter the desired information into the `CM ACIDP/CRDP POINT ASSIGNMENT` display; save the point data in an IDF and then load the point.

If the point is a CRDP, it must have at least one associated "package," i.e., a CDS. For an ACIDP, any CDS is optional.

NOTE

Before deleting an ACIDP from the CG, you should first uninstall its ACP; otherwise, the VAX status table will incorrectly show the ACP still to be connected to its ACIDP. There is no other effect from this and the table is automatically corrected after a restart, or a later uninstall.

4.3.1 ACIDP Scheduling Recommendations

Set long RTPERIOD values for cyclic and periodic ACPs, or set them to demand-only, and determine how long they actually run before selecting the normal running period.

If the RTPERIOD in an ACIDP is short (close to the time required for the associated ACP to execute), it will be difficult or impossible to disconnect the ACIDP or to uninstall the ACP. If this happens, change the parameter INH_STAT to INHIBIT from the point's Detail Display at a Universal Station. Wait for the ACP to terminate as indicated by a change of Execution State to DELAY, then disconnect the ACP from the ACIDP.

4.3.2 The ACCESSKEY Parameter

In order for an ACP to be able to store data to points on the LCN or to send messages, these conditions must be met:

- The ACP must be connected to an ACIDP.
- The ACP installation mode must be NORMAL.
- The ACIDP parameter ACCESSKEY value must equal READWRIT.

If the ACP is intended neither to store data on the LCN, nor to send messages, the ACCESSKEY value of its ACIDP should be set to READONLY.

4.3.3 Support for Continuous Control

To allow direct process control from an ACP, the RCASENB parameter of the connected ACIDP must be set to ON. This parameter provides the capability to enable continuous (AM like) control to a slot on a process connected box (Hiway or UCN devices). Continuous or cascade control implies that the process connected slot/point will fail to its local backup control mode if a SP or OP store is not performed within the time span of the slot's shed timer (BOXTOGn/SHEDTIME). The SP or OP store is performed by an ACP program in an upper level processor (e.g., a CM50S) which is attached to an ACIDP point in the CG.

When this parameter's value is ON, continuous control is enabled; when its value is OFF, control reverts to the normal control state which is program control.

To properly use continuous control, the ACP program must conform to these continuous control rules:

- The process connected box and Hiway/UCN must be in FULL control.
- The ACP establishes cascade to the process-connected slot by storing the value of Cas to the MODE parameter when the ACP detects that the CASREQ parameter's value is Request.
- The ACP program should delay for at least one second after storing the MODE parameter value before storing the SP or OP parameter (this allows the process connected slot to initialize).
- The ACP needs to make use of secondary point parameters that reside in the process connected slot. Some of the important secondary point parameters are:

  - CASREQ
  - MODE and MODATTR
  - SP and OP
  - INITVAL and INITMAN
  - TRACK
  - RCASENB/RCASOPT
  - PVEUHI and PVEULO
  - SPEUHI and SPEULO
  - ARWNET and ARWOP
  - RCASSHED

ACIDPs that will **not** be connected to an ACP that directly controls process operations should be configured with the parameter RCASENB (Remote CAScade ENaBle) set to OFF to prevent the accidental storing of SP and OP parameter values.

4.4 ACIDP AND CRDP DISPLAYS

The current values of certain ACIDP and CRDP parameters can be viewed—and some scheduling-related parameters can be changed—from the area Universal Station in the Operator Personality. Some of the ACIDP parameters also can be viewed —but not changed—from a VAX terminal through the ACIDP Detail display (reached through the CG/PLNM Database displays). See heading 4.5.5 for ACIDP Detail display information.

Display and change access to ACIDP parameters is summarized in Table 4-1. Display access to CRDP parameters is summarized in Table 4-2.

Brief descriptions of user-visible ACIDP and CRDP parameters follow at paragraph 4.4.1. The definitions also include the parameter names as shown in ACIDP Detail display.

Table 4-1 — Display and Change of ACIDP Standard Parameters

PARAMETER NAME	UNIVERSAL STATION DETAIL DISPLAY		UNIVERSAL STATION GROUP DISPLAY		VAX ACIDP DISPLAY
	Display	Change	Display	Change	Display
ABORTCOD	no*	no	no	no	no
ACCESKEY	yes	no	no	no	yes
ACPROG	yes	no	no	no	yes
ACT_TYPE	yes	no	no	no	yes
CONF_RQD	no	no	no	no	yes
CONFWAIT	yes	no	no	no	yes
EXECSTAT	yes	no	yes	no	yes
INH_STAT	yes	operator	no	no	yes
KEYWORD	yes	no	yes	no	yes
NAME	yes	no	yes	no	yes
NEXT_RTM	yes	no	no	no	yes
OPER_DMD	yes**	operator	no	no	no
PROGSTAT	yes	no	no	no	yes
PTDESC	yes	no	yes***	no	yes
RCASENB	yes	yes	no	no	no
RTPERIOD	yes	no	no	no	yes
RUN_INIT	yes	no	no	no	yes
STIME	yes	no	no	no	yes
TAKE_I_P	no	no	no	no	yes
UNIT	no	no	no	no	yes

* Visible as EXECSTAT value when nonzero
** Target used to request ACP activation
*** Only shown when point is selected

Table 4-2 — Display of CRDP Parameters

PARAMETER NAME	UNIVERSAL STATION DETAIL DISPLAY		UNIVERSAL STATION GROUP DISPLAY	
	Display	Change	Display	Change
KEYWORD	yes	no	yes	no
NAME	yes	no	yes	no
PTDESC	yes	no	no	no
UNIT	yes	no	no	no

4.4.1 CG/PLNM Parameter Descriptions

The following are brief descriptions of the user-visible ACIDP and CRDP standard parameters. For details, see the *Computer Gateway Parameter Reference Dictionary.*

ABORTCOD—Four-character code that indicates the reason for abort of the ACP. See the ACP Termination Interface (heading 11.1.5, 15.1.5, or 19.1.5) for an explanation of abort code assignments. When not zero, this value replaces EXECSTAT value in the US Detail display.

ACCESSKEY—Determines whether or not the ACP can execute writes to the LCN. Values are READWRIT or READONLY. (Shown by ACIDP Detail display as `LCN Access.`)

ACPROG—The abbreviated pathname (bound-unit name) of the ACP as stored in the VAX. (Shown by ACIDP Detail display as `ACP Name.`)

ACT_TYPE—The activation method for the ACP. Values are CYCLIC, PERIODIC, CYC_DMD, PER_DMD, DEMAND. (Shown by ACIDP Detail display as `Point Schedule.`)

CONF_RQD—Set ON when an operator message requiring confirmation is sent. Must be true before a message confirmation is processed. (Shown by ACIDP Detail display as `Waiting for Message Confirm.`)

CONFWAIT—Time (in seconds) remaining before a pending message confirmation times out. (Equal to zero when CONF_RQD equals OFF.)

EXECSTAT—The present execution state of the ACP. Values are ABORT, ACCESS, DELAY, OFF, RUN, WAIT, FAIL. When state is ABORT, the Universal Station Detail display shows the current value for ABORTCOD. (Shown by ACIDP Detail display as `Execute State.`) Note that the CG/PLNM is unaware of the CM50S ACP operating states of Hibernate and Delay, and continues to show the ACP state as RUN.

INH_STAT—An operator-changeable parameter that controls activation of the ACP. Values are INHIBIT or PERMIT. (Shown by ACIDP Detail display as `ACP Activation.`)

KEYWORD—Optional point descriptor shown on ACIDP displays.

NAME—Name of the ACIDP/CRDP point. (Shown by ACIDP Detail display as `Point Name.`)

NEXT_RTM—Next runtime, used by both Periodic and Cyclic activation. The parameter is in form HH:MM:SSΔMM:DD:YYΔ (where Δ indicates a space). Blank if activation type is demand-only. (Shown by ACIDP Detail display as `Next Run Time.`)

OPER_DMD—An operator-accessible parameter, which when set ON, turns on the ACP if its activation type permits demand activation and ABORTCOD = 0. (See the *Computer Gateway Parameter Reference Dictionary* for effect of OPER_DMD on an Aborted ACP.)

PROGSTAT—Installation mode of the ACP. Values are NOT INST, TEST, RESTRICT, NORMAL. (Shown by ACIDP Detail display as `ACP Status.`)

PTDESC—Description of the variable. (Shown by ACIDP Detail display as `Description.`)

RCASENB—This is the Remote Cascade Enable Flag which can be used to enable continuous control through a process connected box on a Data Hiway or UCN on a connected LCN. Continuous or cascade control implies that the process connected slot/point will fail to its local backup control if a SP or OP store is not performed by an ACP within the time span of the slot's shed timer. When the value of this parameter is ON, continuous control is enabled; OFF reverts to normal control.

RTPERIOD—The time period between runs of a scheduled ACP, in format HH:MM:SS. Minimum period is 10 seconds; maximum period is 24 hours. Not used if ACP activation is demand-only. (Shown by ACIDP Detail display as `Period.`)

RUN_INIT—When ON, tells the scheduler to turn on the ACP immediately after an "initialization event" (see note below). (Shown by ACIDP Detail display as `Run at CG Initialize.`)

STIME—The first time of day that a periodic program runs, in format HH:MM:SS. The maximum time is 24:00:00. Not used if ACP activation is cyclic, cyclic-demand, or demand. (Shown by ACIDP Detail display as `Start Time.`)

TAKE_I_P—Set ON at every "initialization event" (see note below) to inform the ACP to take its initialize path. Also can be set ON by CL programs and ACPs. (Shown by ACIDP Detail display as `Take Initialize Path`)

UNIT—Unit identification number.

NOTE

There are three initialization event types:

1. CG/PLNM power up and software load (cold restart)
2. CM50S initialization or HDLC data-link restart (warm restart)
3. ACIDP initialization
   a. Connecting an ACP using ACP Operations displays
   b. Removing an Abort condition from an ACP

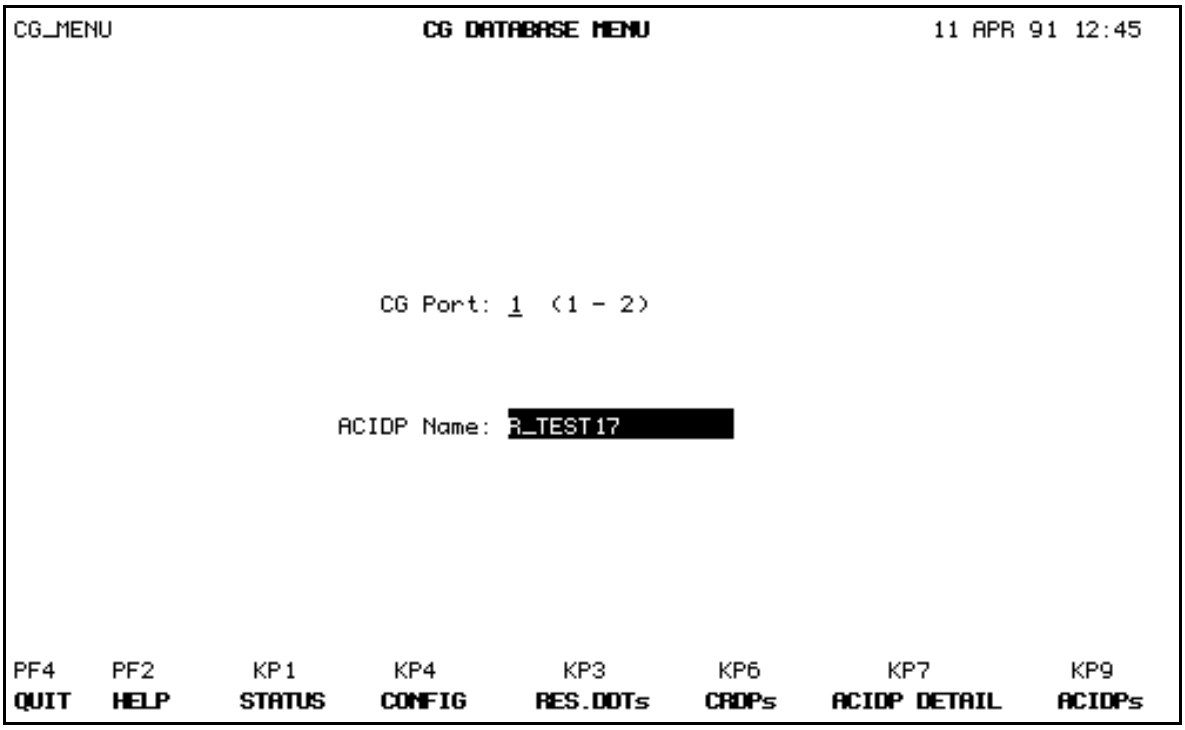
4.5 CG/PLNM DATABASE DISPLAYS

The CG/PLNM Database displays are activated from the CM50S Main Menu (see paragraph 2.8). The hierarchy of displays for the CG/PLNM Database displays is

CG Database Display
- Status of Computer Gateways—KP1
- Resident DDT Summary—KP3
- Calculated Results Data Points—KP6
- ACIDP Detail—KP7
- ACIDP Summary—KP9

4.5.1 CG Database Display

All major CG/PLNM Display Operations are invoked from the CG Database Display screen. Lists of CG/PLNM Resident DDTs, CRDPs, and ACIDPs can be viewed. Other options show the link status of all CGs connected by the CM50S configuration procedure as well,and the parameter values for a specified ACIDP.



4.5.1.1 CG Database Display Entry Fields

- `CG Port Number`—This field indicates the number of the CG Port to be used for data requests. This must be a number from 1 to 4. The default value is `1`. Entry in this field is required for all functions on this screen except for `Help` and `Status`.

- `ACIDP Name`—This is the 1-to-16-character name of the ACIDP to be viewed. Entry in this field is required for only the `KP7/ACIDP DETAIL` function.

4.5.1.2 CG Database Display Function Keys

PF4/QUIT—Exits CG Database Display screen and returns to the CM50S Main Menu.

PF2/HELP—See Heading 3.1.3.

KP1/STATUS—Calls up the Status of Computer Gateways display. See heading 4.5.2.

KP3/RES DDTS—Lists the names of all CG-Resident DDTs for the CG identified in the CG Port Number field. See heading 4.5.3.

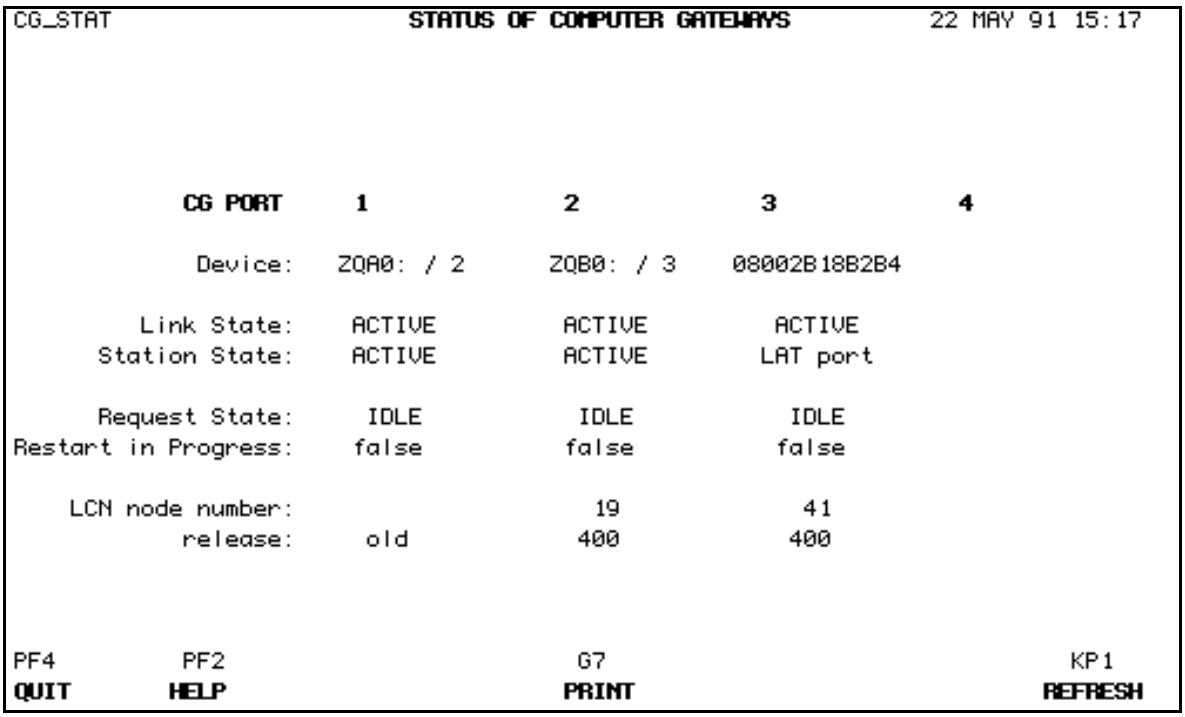
KP4/CONFIG—Displays the LCN configuration parameters for the CG identified in the CG Port Number field. This function works only if the CG is running TDC 3000 release 400 or later. See heading 4.5.7.

KP6/CRDPs—Lists the names of all CRDPs (Calculated Results Data Points) for the CG identified in the CG Port Number field. See heading 4.5.4.

KP7/ACIDP DETAIL—Brings up the detail display for the Advanced Control Data Interface Point identified in the ACIDP Name and CG Port fields. See heading 4.5.5.

KP9/ACIDPs—Lists the names of all ACIDPs for the CG identified in the CG Port Number field. See heading 4.5.6.

4.5.2 Status of Computer Gateways Display



4.5.2.1 Status of Computer Gateways Display Fields

- `CG Port`—The logical connection number (1-4) that identifies a CG to the CM50S software. This number is assigned by the CM50S configuration program and is not related to the CG's node number on the LCN.

- `Device`—For a PLNM connection, this displays the last 5 hexidecimal characters of the ethernet address. For an HDLC connection, this displays the physical device name/slot number of the HDLC I/O board on the VAX.

- `Link State`—Identifies the state of the I/O communications board. The possible values are

  ACTIVE - normal operation
  INACTIVE - logically unavailable (probably software initialization underway)
  DISABLED - electrically disconnected

- `Station State`—Identifies the state of the specific I/O port. The possible values are

  ACTIVE - normal operation
  INACTIVE - logically unavailable (suspect CG is powered down)
  DISABLED - electrically disconnected (suspect loose cable)
  LAT PORT - I/O is configured to use the LAT protocol over ethernet

- Request State—Identifies current message activity over the link. The possible values are

  IDLE - no message being processed
  TRANSMIT - active I/O, a message is being transmitted or received

  CONFIRM - the VAX is waiting for a confirmation message from the CG
  SEGMENT - a large transaction is being assembled/disassembled into segmented message packets for transmission

- Restart in Progress—FALSE during normal operation; CG database is synchronized with the VAX. TRUE means either that the CG has signaled a link reset and its database is being re-synchronized with the VAX, or that the CG has remained down since the CM50S software was started (no synchronization signal has been received).

- LCN node number—Identifies the node number of the CG within its LCN. This value is displayed only if the LCN software is TDC 3000 release 400 or later.

- Release—Identifies the release of TDC 3000 software running on the LCN. This value is displayed only if the LCN software is TDC 3000 release 400 or later.

4.5.2.2 Status of Computer Gateways Display Function Keys

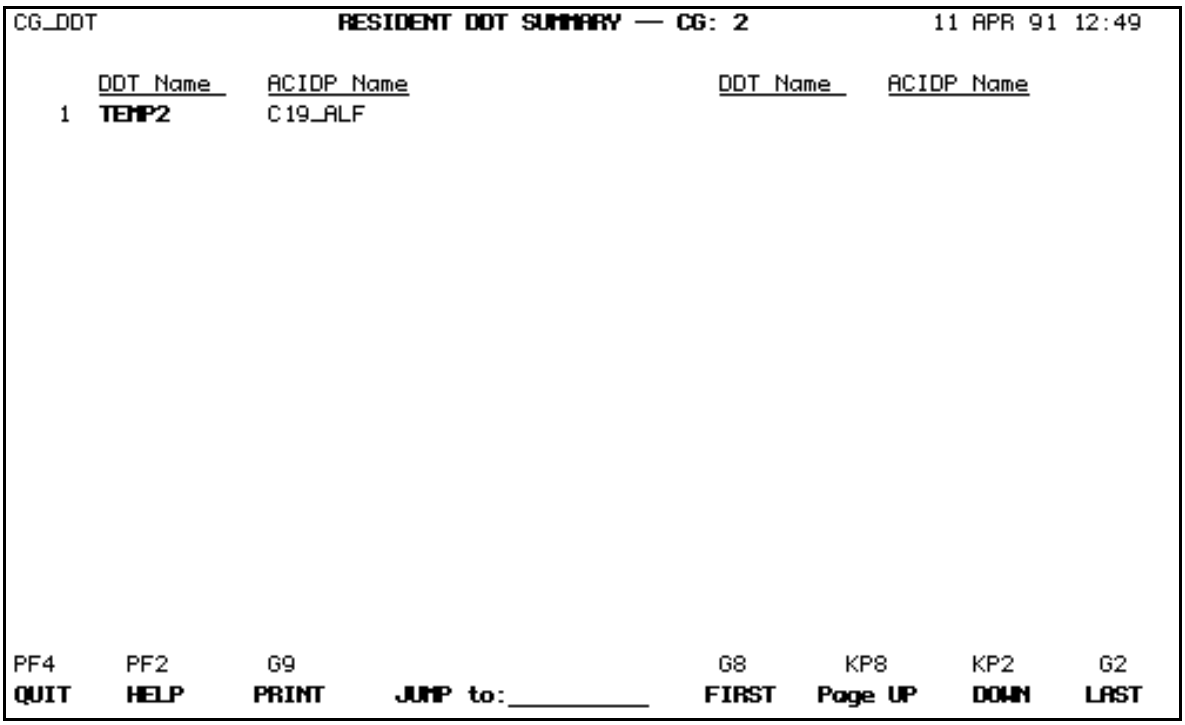
PF4/QUIT—Exits Status of Computer Gateways Display screen and returns to the CM50S Main Menu.

PF2/HELP—See heading 3.1.3.

G7/PRINT—Causes the Status of Computer Gateways display to be printed to the default printer.

KP1/REFRESH—Causes the Status of Computer Gateways display to be refreshed on the screen.

4.5.3 Resident DDT Summary Display



4.5.3.1 Resident DDT Summary Data Display Fields

This screen lists all of the DDTs that are installed in the specified CG. Up-to-30 entries can be displayed at a time. If the DDT is connected to an ACIDP, the ACIDP name appears next to the DDT name.

4.5.3.2 Resident DDT Summary Function Keys

PF4/QUIT—Cancels viewing of the List CG-Resident DDTs Display and returns to the CG Database Display.

PF2/HELP—See heading 3.1.3.

G9/PRINT—Prints the entire list of CG resident DDT names to the default printer.

JUMP TO—Type the entry number or the name of the DDT to be displayed and press <RETURN>. The line containing that DDT name then is displayed as the first data line on the screen.

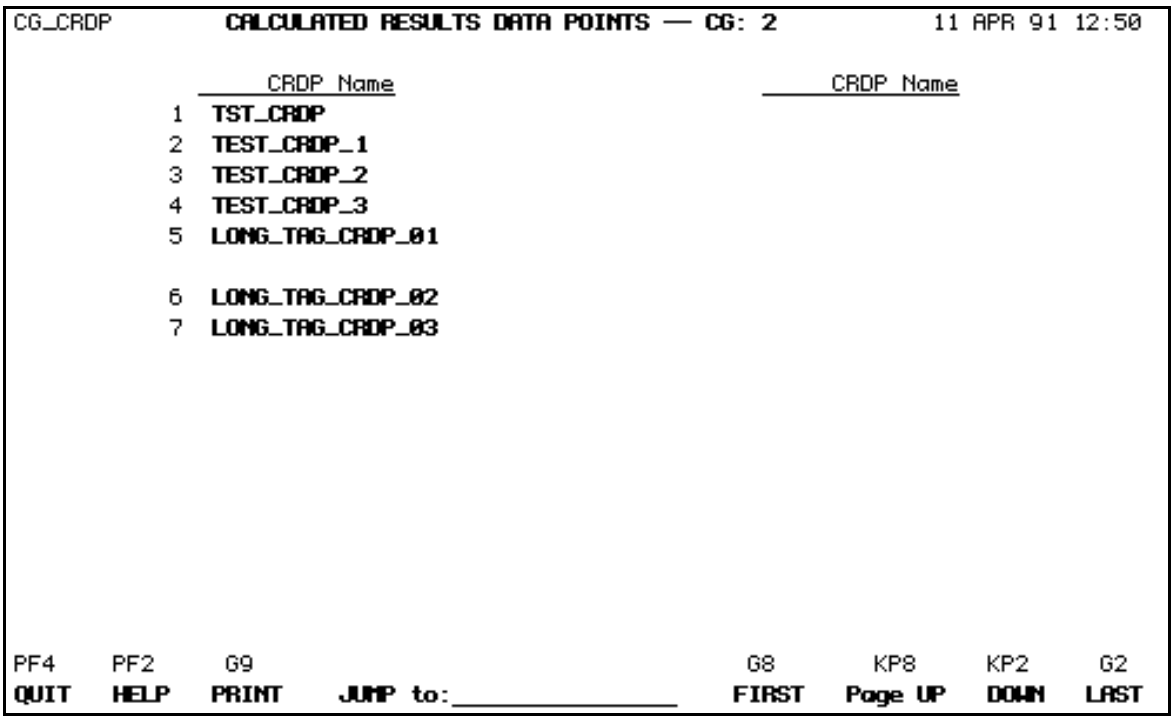
G8/FIRST—Displays the first page of the DDT List. If the first page of entries is already on display, a message stating so appears at the bottom of the screen.

KP8/PAGE UP—Displays the previous page of DDT entries. If the first page of entries is already on display, a message stating so appears at the bottom of the screen.

KP2/PAGE DOWN—Displays the next page of DDT entries. If the last page of entries is already on display, a message stating so appears at the bottom of the screen.

G2/LAST—Displays the last page of the DDT entries. If the last page of entries is already on display, a message stating so appears at the bottom of the screen.

4.5.4 Calculated Results Data Points Display



4.5.4.1 Calculated Results Data Points Display Fields

This screen lists the names of all Calculated Results Data Points that can be accessed through the specified CG. The data is displayed in two columns, and up-to-30 CRDP names can be displayed on the screen at a time.

4.5.4.2 Calculated Results Data Points Display Function Keys

PF4/QUIT—Cancels viewing of the List CRDPs Display and returns to the CG Database Display screen.

PF2/HELP—See heading 3.1.3.

G9/PRINT—Prints the entire list of CRDP names to the default printer.

JUMP TO—Type the entry number or the name of the CRDP to be displayed and press <RETURN>. The line containing that CRDP name is then displayed as the first data line on the screen.

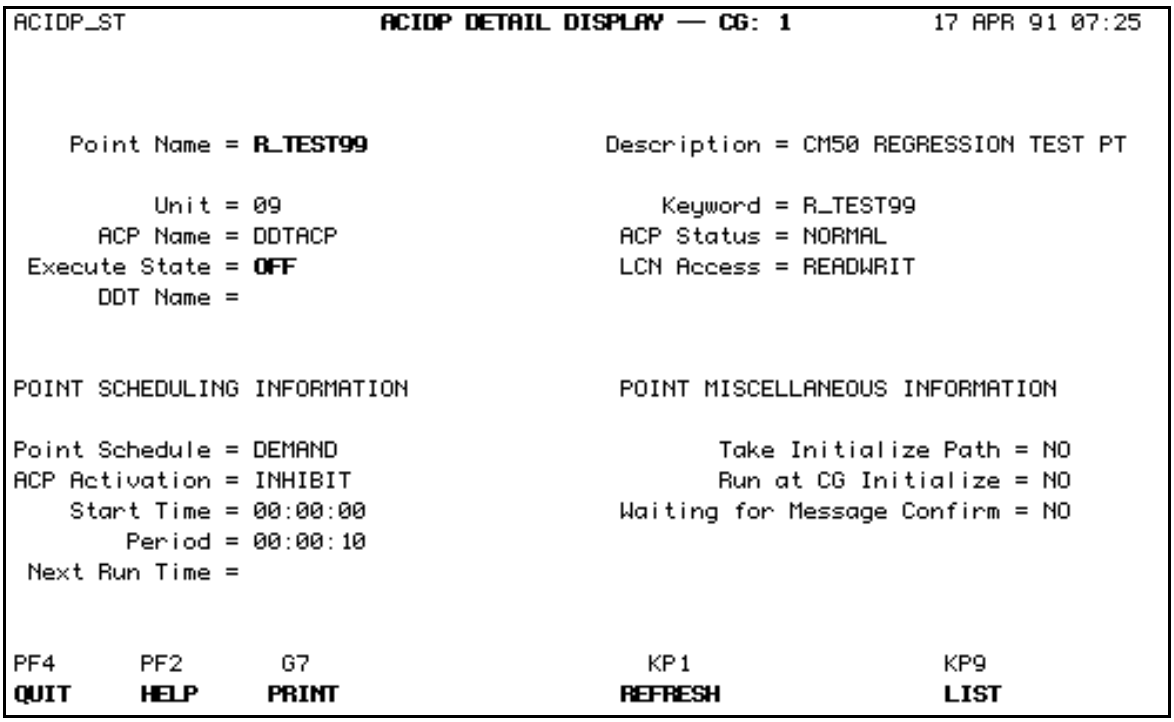
G8/FIRST—Displays the first page of the CRDP list. If the first page of entries is already on display, a message stating so appears at the bottom of the screen.

KP8/PAGE UP—Displays the previous page of CRDP entries. If the first page of entries is already on display, a message stating so appears at the bottom of the screen.

KP2/PAGE DOWN—Displays the next page of CRDP entries. If the last page of entries is already on display, a message stating so appears at the bottom of the screen.

G2/LAST—Displays the last page of the CRDP entries. If the last page of entries is already on display, a message stating so appears at the bottom of the screen.

4.5.5 ACIDP Detail Display



4.5.5.1 ACIDP Detail Display Fields

For a full explanation of all the parameters shown on this display, see the *Computer Gateway Parameter Reference Dictionary*. For a summary explanation, see heading 4.4.1 in this document. The following list equates ACIDP Detail display field names to formal CG Parameter Names used elsewhere.

`Point Name`—NAME
`Unit`—UNIT
`ACP Name`—ACPROG
`Execute State`—EXECSTAT
`DDT Name`—The name of a CG-resident DDT connected to this ACIDP.
`Point Schedule`—ACT_TYPE
`ACP Activation`—INH_STAT
`Start Time`—STIME
`Period`—RTPERIOD
`Next Run Time`—NEXT_RTM
`Description`—PTDESC
`Keyword`—KEYWORD
`ACP Status`—PROGSTAT
`LCN Access`—ACCESKEY
`Take Initialize Path`—TAKE_I_P
`Run at CG Initialize`—RUN_INIT
`Waiting for Message Confirm`—CONF_RQD

4.5.5.2 ACIDP Detail Display Function Keys

PF4/QUIT—Cancels viewing of the ACIDP Detail display and returns to the CG Database display.

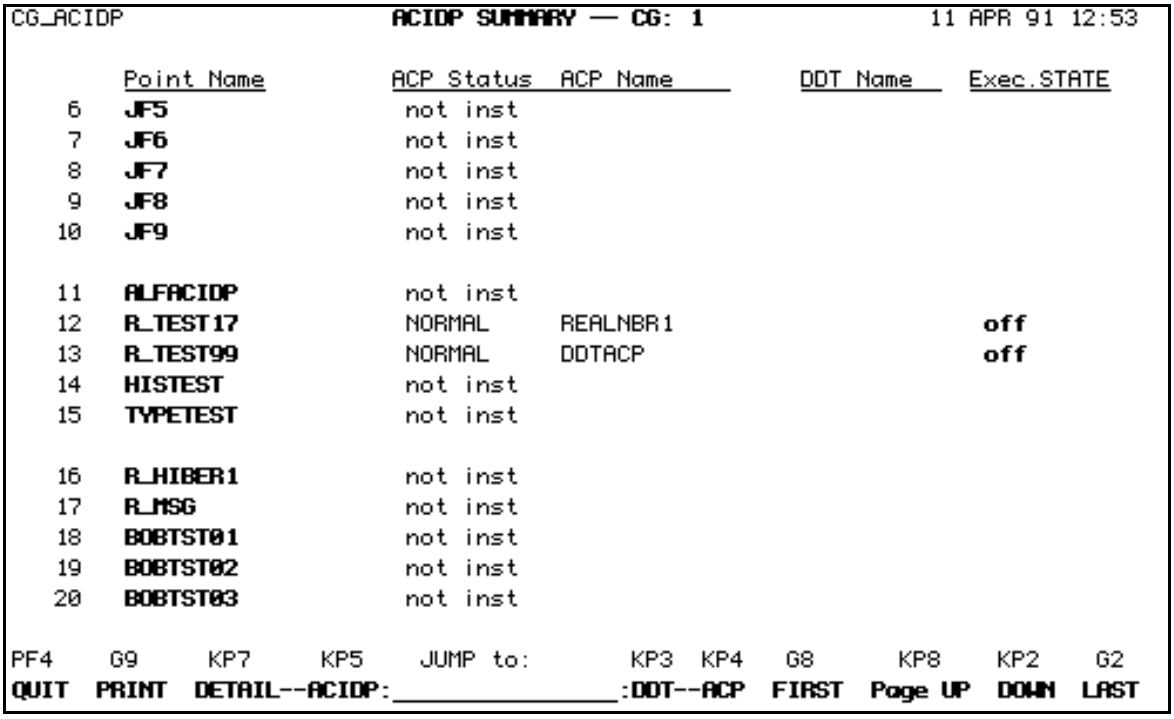
PF2/HELP—See heading 3.1.3.

G7/PRINT—Prints the ACIDP Detail display to the default printer.

KP1/REFRESH—Refreshes the display with any changes that have been made since the screen was first displayed (or last refreshed).

KP9/LIST—Calls up the ACIDP Summary display at its first page.

4.5.6 ACIDP Summary Display



4.5.6.1 ACIDP Summary Display Fields

This screen lists all of the existing ACIDPs on the TDC 3000 that can be accessed through the specified CG. Next to each ACIDP name are shown the ACP status, any associated ACP and (CG-resident) DDT names, and execution status. Up to 15 ACIDP names can be displayed on the screen at once. Note that this display is not dynamically updated, thus any changes to ACP Status or Execution State that occur after the list is requested are not shown.

4.5.6.2 ACIDP Summary Display Function Keys

PF4/QUIT—Cancels viewing of the ACIDP Summary and returns to the CG Database Display screen.

PF2/HELP—See heading 3.1.3.

G9/PRINT—Prints the entire list of ACIDP names to the default printer.

KP7/DETAIL—Type the entry number or the name of the ACIDP to be displayed and press KP7. The ACIDP Detail display for the named ACIDP appears on the screen. See heading 4.5.5.

KP5/ACIDP—Type the entry number or the name of the ACIDP to be displayed and press KP5. The line containing that ACIDP name is then displayed as the first data line on the screen.

KP3/DDT—Type the name of the DDT to be displayed and press KP3. The line containing that DDT name is then displayed as the first data line on the screen.

KP4/ACP—Type the name of the ACP to be displayed and press KP4. The line containing that ACP name is then displayed as the first data line on the screen.

G8/FIRST—Displays the first page of ACIDP entries. If the first page of entries is already on display, a message stating so appears at the bottom of the screen.

KP8/PAGE UP—Displays the previous page of ACIDP entries. If the first page of entries is already on display, a message stating so appears at the bottom of the screen.

KP2/PAGE DOWN—Displays the next page of ACIDP entries. If the last page of entries is already on display, a message stating so appears at the bottom of the screen.

G2/LAST—Displays the last page of ACIDP entries. If the last page of entries is already on display, a message stating so appears at the bottom of the screen.

4.5.7 LCN Configuration Display

If the CG is running TDC 3000 release 400 or later, then the following display is available to view LCN configuration values:

```
CG_CONF              CONFIGURATION DISPLAY — CG: 2              11 APR 91 12:55


LCN: Pinid       = CG                                TDC3000 release = 400
     Max. tag    = 16 characters


 CG: Node #      = 19                                          version =  40.11
     Descr.      = CONNECTED TO THE VAX VIA CABLE B - 68020

     Speed       = 56700 baud                        Time synch.     =  15
     Station     = 1                                 Confirm Time    =  10
     Float point = IEEE                              T1 Time         =  14
                                                     N2 Time         =   3


 HM:   User Aver =  5 min.                           Month type      = Calendar
     Shifts/week =  21                               Week start Hr   =   0
                                                          (0 = start of Sun.)


PF4        PF2                  G7                                   KP1
QUIT       HELP                 PRINT                              STATUS
```

4.5.7.1 LCN Configuration Display Fields

LCN System-Wide Values:

- Pinid—The 1 or 2 character LCN identifier for Network Gateway routing.
- TDC 3000 release—The software release level running on the LCN.
- Max. tag—The maximum length of points on the LCN (either 8 or 16 characters).

CG Configuration Parameters:

- Node #—The node number of this CG on the LCN
- version—The CG personality release and revision levels (separated by a period).
- Descr.—The Description Parameter for the CG
- Speed—The communications baud rate configured at the CG
- Station—The communications station number of the CG. (This should be 1 or 3 and **not** match the station number of the CG port on the VAX.)
- Float point—The format of floating point numbers received from the LCN. This should be "IEEE".
- Time synch.—The maximum time between communications transactions sent by the CG. Normal is 15 seconds.
- Confirm time—The time out limit for the CG to wait for a message confirmation. Normal is 2 seconds.
- T1 Time—The CG's T1 time out counter. Normal is 14.
- N2 Time—The CG's retry counter. Normal is 3.

History Module configuration values:

- User Aver.—The number of minutes for User Averages.
- Shifts/week—The number of shifts per week.
- Month type—The type of monthly averages, either Calendar month or 28-day fiscal months.
- Week start Hr.—The starting hour of the week for averages. 0 is the start of Sunday, 12 is noon Sunday, 24 is the start of Monday, etc.

4.5.7.2 LCN Configuration Display Function Keys

PF4/QUIT—Exits from the display and returns to the main CG Database display.

PF2/HELP—See heading 3.1.3.

KP1/STATUS—Exits from the display and jumps to the CG Status display.

G7/PRINT—Prints the LCN Configuration display to the default printer.

PROGRAM INSTALLATION AND TESTING
Section 5

This section explains the process of application program installation, testing, and revision.

5.1 PROGRAM LINKING

References to several libraries are required to link your application programs. The DCL command file named ACP_LINK is provided to supply those references. To link an application program and place it in the default ACP directory (CM50$ACP:), enter the following command at the DCL prompt:

`$ @CM50$LIB:ACP_LINK acp_name` (where `acp_name` is any user-written program)

5.2 ACP OPERATIONS SCREEN

All major ACP operations such as `INSTALL`, `CONNECT`, `ACTIVATE`, and `LIST` are invoked from the ACP Operations screen. This screen is accessed through the CM50S Main Menu.

```
ACPOPER                          ACP OPERATIONS                  11 APR 91 14:05


    ACP Name: MPLT30_____                  Execution State = OFF

ACP Pathname:
$1$DUS1:[CM50S.R_TEST]MPLT30.EXE

          Installation Mode: N                 ACIDP Name: C19_DICK_______
             N = Normal                           CG Port: 2  (1-4)
             T = Test
             R = Restricted                      DDT Name =


                        Activation/Deactivation Mode: R
                         I = Interactive  (Activate)
                         R = Remote       (Activate)
                         O = Off          (Deactivate)
                         A = Abort        (Deactivate)


PF4       G0         G.        KP0        KP.          G3          KP9      PF3
QUIT    INSTALL    UNINST   ACTIVATE    DEACT   MOD CONNECTION    LIST     MORE
```

The hierarchy of displays that start from the ACP Operations screen is:

ACP Operations
- Modify Program Connection with ACIDP—G3
- ACP List—KP9
- ACP Installation (edit)—KP1
- Installer Activity Log—KP4

5.2.1 ACP Operations Screen Fields

- `ACP Name`—This is the 1 to 12 character name of the ACP. This field must be filled in for all cases except Quit, List, Display Log, Print All and More. After the ACP name has been entered, a user action routine checks to see if the ACP has already been installed. If the ACP is installed, then all pertinent fields on the screen are automatically filled in to show the existing parameters. The cursor is positioned on the DDT Name field. At this time, modifications can be made to any field on the screen.

- `Execution State`—Display only. Shows current execution state of the ACP at the time the screen was called up. See heading 2.2.2.8.

- `ACP Pathname`—This is the full (up to 80 characters) pathname of the ACP executable file. This field allows operations to be performed on ACPs that do not reside in the standard CM50$ACP directory, and also allows the ACP name to be different than the ACP executable file name. If the ACP is already installed, then this field is filled in automatically. If this field is left blank, the pathname defaults to the "CM50$ACP:" directory with a file name identical to the ACP name, with an extension of ".EXE". Optionally, the pathname can include a file version number. If an ACP is installed without specifying a file version number, then each activation will use the current version of the executable file, even if it has been modified after the installation. Note that the pathname is dynamically expanded to specify the volume and extension names, and that the 80-character length maximum applies to the expanded pathname.

- `Installation Mode`—This field is used to select or change ACP installation mode. For a currently installed ACP, the existing mode selection is shown. Refer to heading 2.2.2.7 for information on the three ACP installation modes.

- `ACIDP Name`—The name (up to 16 characters) of an ACIDP that the ACP is to be connected to. This is an optional entry. If the ACP is already installed with this parameter entered, then this field is filled in automatically.

- `CG Port`—This field identifies the port number of the CG that is to contain the ACIDP. This must be a number from 1 to 4. The default is 1. Entry in this field is required only for the Install and Modify Connection functions.

- `DDT Name`—Display only. Appears only when the ACIDP has been set up for data prefetch (see heading 6.4).

- `Activation/Deactivation Mode`—When activating an ACP, enter I for Interactive or R for Remote. When deactivating an ACP, enter O for Off/Delay or A for Abort.

Interactive activation of an ACP runs the program as a subprocess, with the current user's privileges and quotas, and allows you to use the terminal to interact with the program. Interactive mode must be selected if the ACP has been linked with the VMS DEBUG utility.

Remote activation runs the ACP as a detached process with the privileges and quotas specified during ACP installation.

The deactivate feature provides an orderly shutdown regardless of ACIDP connection or installation mode. It is useful to terminate an ACP caught in an endless loop or to change an ACP's Execution State (EXECSTAT) at the ACIDP from Abort to Off or Delay.

5.2.2 ACP Operations Screen Function Keys

The menu on the bottom of the screen displays the names of the function keys that can be pressed to perform ACP functions such as installing, uninstalling, activating, or deactivating an ACP. And also the making or breaking of an ACP/ACIDP connection. Each function on this menu is described below.

PF4/QUIT—Exits ACP Operations and returns to the CM50S Main Menu.

G0/INSTALL—Installs the ACP and adds it to the ACP Table. All fields except for the Activation/Deactivation Mode field are used for ACP installation; however, the ACP Pathname, and ACIDP Name fields are optional entries. Default values are used for all of the additional installation parameters (process name, user identification code, SYS$INPUT, SYS$OUTPUT, SYS$ERROR, privileges, process quotas and status flags) described at heading 5.5. On completion, either an installation complete message or an error message is displayed at the bottom of the screen.

G./UNINST—Uninstalls the ACP and removes it from the ACP Table. The ACP Name field must contain the name of the ACP to be uninstalled. If the ACP name is invalid, then an error message is displayed at the bottom of the screen. Entries in all other fields are ignored.

KP0/ACTIVATE—Activates the ACP according to the contents of the ACP Name and Activation/Deactivation Mode fields. If the ACP name or the activation mode is invalid, then an error message is displayed at the bottom of the screen. Entries in all other fields are ignored. Note that remote activation creates a detached process similar to triggering the ACIDP, while interactive activation runs the ACP as a subprocess using the terminal as SYS$INPUT and SYS$OUTPUT.

KP./DEACTIVATE—Deactivates (forces termination of) the ACP according to the contents of the ACP Name and Activation/Deactivation Mode fields. If the ACP name or the deactivation mode is not valid, then an error message is displayed at the bottom of the screen. Entries in all other fields are ignored.

Note that deactivating an ACP issues a VMS STOP process command that kills the ACP without waiting for completion of any pending LCN call. If the ACP is accessing a DDT when it is deactivated, that DDT may be left locked.

To unlock a DDT: RUN CM50$SUPPORT:CM50_CLEAR_DDTUSE. This utility prompts for the DDT name, then informs the user of its success.

G3/MOD CONNECTION—Causes the MODIFY PROGRAM CONNECTION WITH ACIDP screen to be displayed. Refer to heading 5.3 for additional details.

KP9/LIST—Displays all of the installed ACP names and other pertinent information on the screen. This includes each ACP's name, source path, installation status, ACIDP name, execution state, CG port, and process identification number. The entries in all fields will be ignored. The user can page forward and backward through the ACP names. Refer to heading 5.4 for more information.

PF3/MORE—Displays the alternate function key menu (shown following) at the bottom of the screen.

PF4	PF2	KP1	G1	KP4	G9	PF3
QUIT	**HELP**	**EDIT PARAMS**	**CHG INST MODE**	**DISP LOG**	**PRT ALL**	**MORE**

The additional functions on this menu are described below.

PF2/HELP—See heading 3.1.3

KP1/EDIT PARAMS—Displays the first of two ACP installation screens that allow modification of the default parameters for ACP installation. The parameters that appear on the first installation screen are the process name, user identification code, SYS$INPUT, SYS$OUTPUT, SYS$ERROR and privileges. The second installation screen allows the process priority, quotas, and status flags to be modified, if necessary. Refer to heading 5.5 for detailed information.

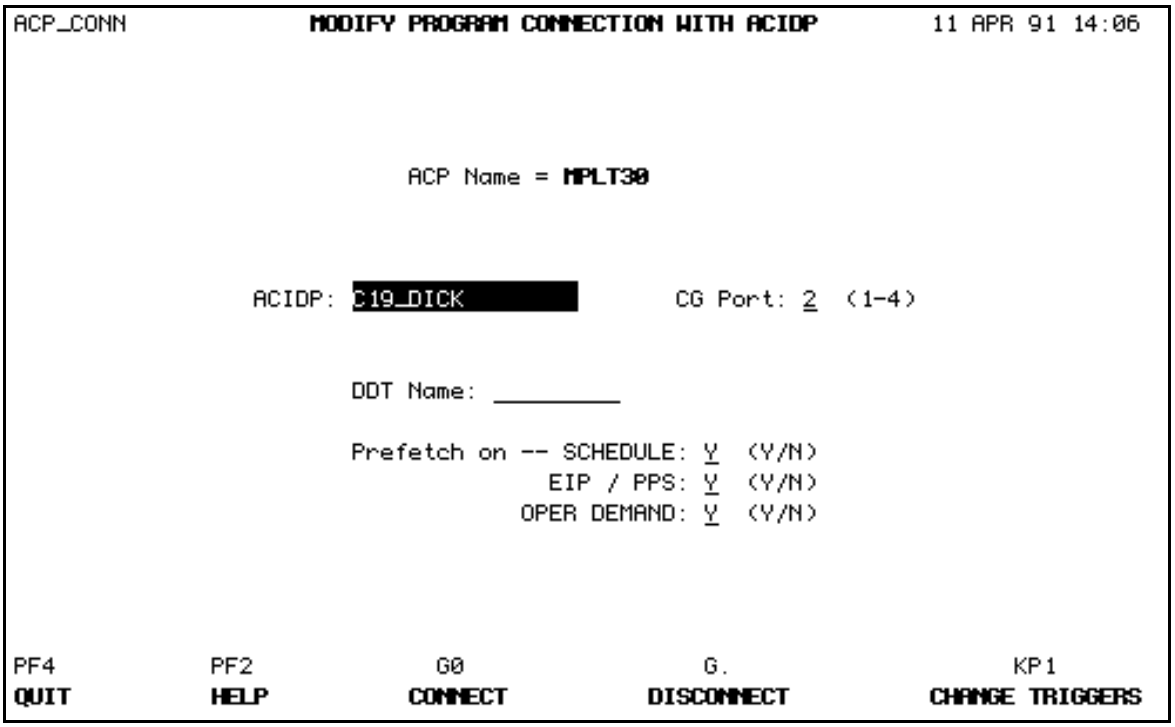
G1/CHG INST MODE—Changes the installation mode of the ACP according to the contents of the ACP Name and Installation Mode fields. If the ACP name is invalid, then an error message is displayed at the bottom of the screen. Entries in all other fields are ignored.

KP4/DISP LOG—Displays the Installer Activity Log which records any significant changes in the status of individual ACPs. Refer to heading 5.6 for detailed information.

G9/PRT ALL—Prints all of the installed ACP names and other pertinent information to the default printer. This includes each ACP's name, source path, installation status, ACIDP name, execution state, CG number, and process identification number. (The same information provided by the ACP List Screen.) The entries in all fields are ignored. After the ACP Table has been sent to the printer, the message “Print complete” appears at the bottom of the screen.

5.3 MODIFY PROGRAM CONNECTION WITH ACIDP SCREEN

This screen is displayed as the result of the G3/MOD CONNECTION selection from the ACP Operations Screen. From this display the user can modify the ACP/ACIDP connection, the ACIDP/DDT connection, and/or the Triggers.



5.3.1 Modify Program Connection With ACIDP Screen Data Fields

- `ACP Name`—This is the up-to-12 character name of the ACP entered at the ACP Operations screen. It cannot be changed from this screen.

- `ACIDP Name`—This is the one-to-eight character name of the ACIDP to connect the ACP to. If the ACP is already installed with this parameter entered, then this field is filled in automatically.

- `CG Port Number`—This field indicates the CG port number to be used for data requests. This must be a number from 1 to 4. The default is `1`.

- `DDT Name`—This is the one-to-nine character name of a DDT to be connected to the same ACIDP as the ACP (for precollection of data). If the ACP is already installed and this parameter has been entered previously, then this field is filled in automatically. Associating a DDT with an ACIDP allows the CG to fetch data for the DDT immediately on occurrence of predefined scheduling events. This minimizes time lag between the EIP and data capture.

- `PREFETCH on`—The three Y/N field selections (SCHEDULE, EIP/PPS, and OPER DEMAND) allow for the selection of up to three types of "triggers" for prefetch of data when the ACP-ACIDP-DDT connections exist. See heading 6.4 for more information on data prefetch.

5.3.2 Modify Program Connection with ACIDP Screen Function Keys

G0/CONNECT—Connects the ACP to the named ACIDP. If the ACP or ACIDP name is invalid or if the ACP is already connected to another ACIDP, an error message is displayed on the bottom of the screen. If the DDT Name field contains a valid DDT name, the DDT is made CG Resident (if necessary) and the DDT is attached to the named ACIDP for pre-collect of data.

G./DISCONNECT—Disconnects the ACP from the ACIDP. If the ACP name is invalid or if the ACP is not connected to an ACIDP, then an error message is displayed at the bottom of the screen. If a DDT is attached to the named ACIDP, that connection also is broken.

KP1/CHANGE TRIGGERS—Changes the Triggers based on the entries in the fields for "Prefetch on."

5.4 ACP LIST SCREEN

When KP9/LIST is chosen from the ACP Operations screen, the ACP List screen is displayed. This screen displays the name of every installed ACP, its full pathname, installation mode, ACIDP name, execution state, CG Port Number, and process identification number. Up to eight entries can be displayed on the screen at one time. Note that this display is not dynamically updated, thus any changes of ACP State that occur after the ACP list is requested are not shown.

```
ACPLIST                              ACP LIST                          11 APR 91 14:07

Name            Mode          ACIDP              DDT          State   CG   PID
DDT30           Normal        C19_ALF                         Off      2
                $1$DUS1:[CM50S.R_TEST]DDT30.EXE
MPLI30          Normal                                        Off
                $1$DUS1:[USER.LINDSEY]MPLI30.EXE
NG_TEST         Normal        CG19A1                          Off      2
                $1$DUS1:[USER.GALBRAITH.NGTESTS]NG_TEST.EXE
TEST_HIBER      Normal                                        Off
                $1$DUS1:[CM50S.R_TEST]TEST_HIBER.EXE
REALNBR1        Normal        R_TEST17                        Off      1
                $1$DUS1:[USER.LINDSEY]REALNBR1.EXE
DDTACP          Normal        R_TEST99                        Off      1
                $1$DUS1:[USER.GALBRAITH.NGTESTS]DDTACP.EXE
NGTESTER        Normal                                        Off
                $1$DUS1:[USER.GALBRAITH.NGTESTS]NG_TEST.EXE
MPLT30          Normal        C19_DICK                        Off      2
                $1$DUS1:[CM50S.R_TEST]MPLT30.EXE

PF4    PF2                                  G9       G8       KP8      KP2      G2
QUIT   HELP    ACP NAME: _____________      PRINT    FIRST    PRIOR    NEXT     LAST
```

5.4.1 ACP List Screen Fields

- ACP Name—The name of an installed ACP.
- Pathname—This is the full pathname of the ACP executable file. The executable filename may be different than the ACP name. This field is never blank.
- Mode—This is the installation mode of the ACP. It contains one of the following keywords: Normal, Test, or Restricted. See heading 2.2.2.7 for more information on ACP installation modes.
- ACIDP Name—This is the name of the ACIDP that the ACP is connected to. This field is blank if no ACIDP connection exists.
- Execution State—This is the execution state of the ACP. It contains one of the following keywords: ABORT, ACCESS, DELAY, FAIL, OFF, PEND, RUN, HIBR, or WAIT. See heading 2.2.2.8.
- CG—A value of 1-4 represents a CG Port Number that the ACP is associated with. A value of zero (0) indicates that the ACP is not connected to an ACIDP.

- PID—This is the VMS process identification number (PID) of the ACP. It is displayed only when the ACP is in ACCESS, DELAY, RUN, HIBR, or WAIT state.

5.4.2 ACP List Screen Function Keys

PF4/QUIT—Cancels viewing of the ACP List and returns to the ACP Operations screen.

PF2/HELP—See heading 3.1.3

ACP NAME—Type the name of an ACP and press <RETURN>. That ACP is then displayed as the first entry on the screen. Pressing <RETURN> without entering an ACP name has no effect on this display.

G9/PRINT—Prints the entire list of ACPs to the default printer.

G8/FIRST—Displays the first page of ACP entries.

KP8/PRIOR—Displays the previous page of ACP entries. If the first page of ACP entries is already on display, a message stating so appears at the bottom of the screen.

KP2/NEXT—Displays the next page of ACP entries. If the last page of ACP entries is already on display, a message stating so appears at the bottom of the screen.

G2/LAST—Displays the last page of ACP entries. If the last page of ACP entries is already on display, a message stating so appears at the bottom of the screen.

5.5 ACP INSTALLATION SCREENS

When the KP1/EDIT PARAMS key is pressed from the ACP Operations screen (with the ACP Name, Installation Mode, and CG Port Number fields filled in), the first of two ACP Installation screens is displayed. Refer to the $CREPRC system service call in the *VAX/VMS System Services Reference Manual* for more information about the fields on the installation screens. All of the fields on either screen are shown with default values.

CAUTION

The default privileges, priority, and quotas are picked up from the VAX Sysgen parameters tuned for your site's hardware and load. They should be overridden only when there is a specific need. Injudicious change of these parameters can result in serious performance problems.

Note that these installation parameters apply only to ACPs running as detached processes. When an ACP is run interactively, the user's terminal, privileges, quotas, etc. are used by the system.

The default values for the process name, user identification code, SYS$INPUT, SYS$OUTPUT, SYS$ERROR, and ACP privileges can be modified from the first ACP Installation screen.

```
ACPINST-1                        ACP INSTALLATION                    11 APR 91 14:08


Process Name: MPLT30            UIC (User Name):

Sys$Input:
_____________________________________________________________________________
Sys$Output:
NL:_________________________________________________________________________
Sys$Error:
NL:_________________________________________________________________________

Privileges:   _ ACNT        _ DIAGNOSE     _ NETMBX      _ PSWAPM      X SYSLCK
              _ ALLSPOOL    _ DOWNGRADE    _ OPER        _ READALL     _ SYSNAM
              _ ALTPRI      _ EXQUOTA      _ PFNMAP      _ SECURITY    _ SYSPRV
              _ BUGCHK      _ GROUP        _ PHY_IO      _ SETPRV      _ TMPMBX
              _ BYPASS      _ GRPNAM       _ PRMCEB      _ SHARE       _ UPGRADE
              _ CMEXEC      _ GRPPRV       _ PRMGBL      _ SHMEM       _ VOLPRO
              _ CMKRNL      _ LOG_IO       _ PRMMBX      _ SYSGBL      _ WORLD
              _ DETACH      _ MOUNT

PF4                      PF2                        G0                      KP2
QUIT                     HELP                       INSTALL                 NEXT
```

5.5.1 ACP Installation Screen 1 Fields

- `Process Name`—This is the up to 15-character name of the process that the ACP will run under. The default is the ACP name that was entered on the ACP Operations screen.

- `UIC (Name)`—This is the up to 12-character name of the VMS account under which the ACP will run. The default is CM50S_MGR. This field can be used to run an ACP under a different group so that it can access resources in that group. Note that in order to install an ACP with an account other than the default or his own name, a user must be registered with VMS privileges (Group or World).

- `Sys$Input`—This is the up to 80-character full pathname of a file from which the ACP will receive input data. If the ACP is run in interactive mode, the default is the terminal.

- `Sys$Output`—This is the up to 80-character full pathname of a file to which the ACP will write output data. If the ACP is run in interactive mode, the default is the terminal; otherwise, the default is the null device (NL:).

- `Sys$Error`—This is the up to 80-character full pathname of a file to which the ACP will write error messages. If the ACP is run in interactive mode, the default is the terminal; otherwise, the default is the null device (NL:).

- `Privileges`—A list of 36 privileges is displayed on this portion of the screen, with a 1-character field to the left of each keyword. The default privilege (SYSLCK) always is displayed with an "`X`" next to it. Type an "X" in the field next to each privilege that the ACP needs during execution and use the space bar to blank out the fields next to the privileges that are not needed. Note that a user must be registered with the VMS privilege before he can assign it to an ACP.

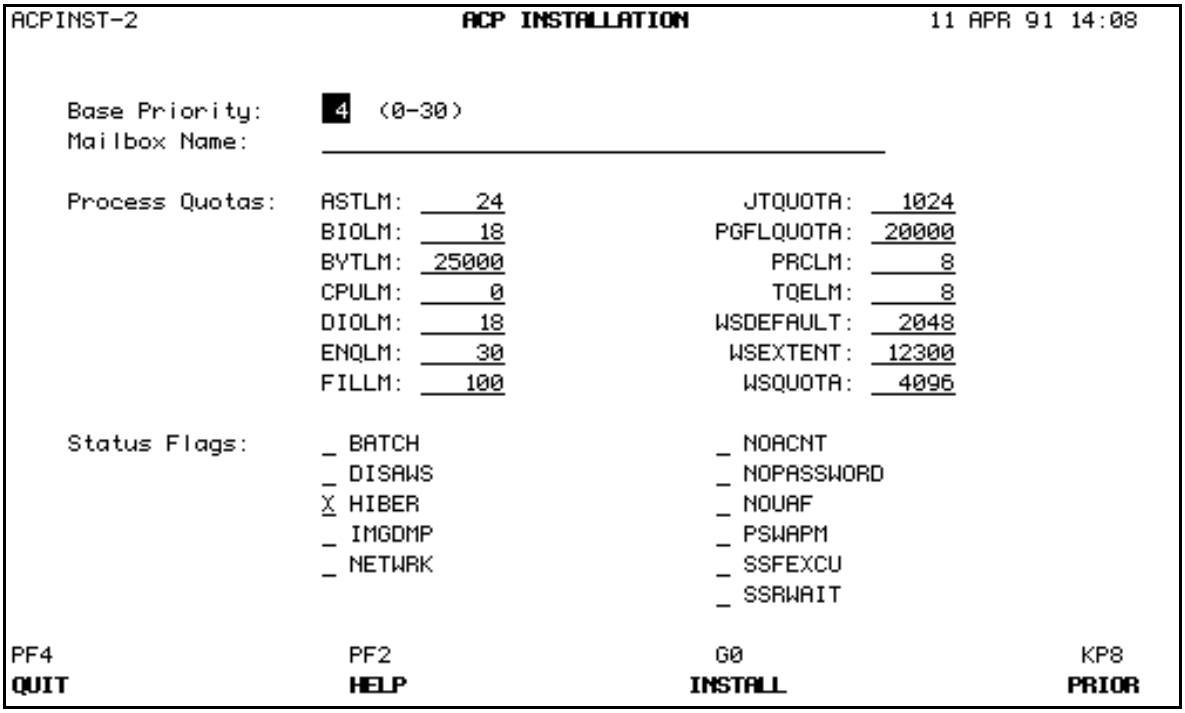
5.5.2 ACP Installation Screen 1 Function Keys

`PF4/QUIT`—Exits ACP Installation and returns to the ACP Operations screen without installing the ACP.

`PF2/HELP`—See heading 3.1.3.

`G0/INSTALL`—Installs the ACP and adds it to the ACP Table. Any newly entered values on the two installation screens (process name, user identification code, sys$input, sys$output, sys$error, and privileges) are used to install the ACP. After the ACP has been installed without errors, the message `ACP has been successfully installed` is displayed at the bottom of the screen. If an error occurs during the ACP installation, an error message is displayed at the bottom of the screen.

`KP2/NEXT`—Displays the second ACP Installation screen. From this screen, the default process priority, quotas, and status flags can be modified before installing the ACP.



5.5.3 ACP Installation Screen 2 Fields

- `Base Priority`—This the runtime priority of the ACP. Enter an integer from 0 to 30, with 0 being the lowest priority. Normal priorities are in the range 0 to 15, and real-time priorities are in the range 16 to 31. The default is 4; caution should be used above that value.

- `Mailbox Name`—40 ASCII characters or less. Used only by applications needing a VMS termination mailbox.

- `Process Quotas`—A list of 14 process quotas is displayed on this portion of the screen, with the default values appearing in the fields. All of the fields can be modified by typing over the existing values.

- `Status Flags`—A list of 13 status flags is displayed on this portion of the screen, with a 1-character field to the left of each keyword. The default flags are displayed with an `X` next to them. Type an `X` in the field next to each desired option and use the space bar to blank out the fields next to options that are not needed.

5.5.4 ACP Installation Screen 2 Function Keys

`PF4/QUIT`—Exits ACP Installation and returns to the CM50S Main Menu without installing the ACP.

`PF2/HELP`—See heading 3.1.3.

G0/INSTALL—Installs the ACP and adds it to the ACP Table. Any newly entered values on the first and second installation screens (process name, user identification code, SYS$INPUT, SYS$OUTPUT, SYS$ERROR, privileges, process quotas, and status flags) are used to install the ACP. After the ACP has been installed without errors, the message `ACP has been successfully installed` is displayed at the bottom of the screen. If an error occurs during the ACP installation, an error message is displayed at the bottom of the second installation screen.

KP8/PRIOR—Returns to the first ACP Installation screen to allow the process name, user identification code, SYS$INPUT, SYS$OUTPUT, SYS$ERROR, and privileges to be modified before installing the ACP.

5.5.5 Use of DCL Within an ACP

There is a significant performance penalty incurred when a process is created with a mapping of the command interpreter. Therefore, access to DCL is not provided to ACPs in the normal remote dispatching of ACPs. For applications that require a combination of ACP functionality (ACIDP triggering and/or LCN stores) and DCL procedures (invoked through the VMS LIB$SPAWN function), the following ACP installation options should be used:

- ACP Pathname – must be specified as: SYS$SYSTEM:LOGINOUT.EXE. This will map the DCL command processor.

- SYS$INPUT – must be installed as the complete file specification of a DCL command file. Example: CM50$ACP:myacp.com. This command file should consist of the following two lines:

```
$ RUN CM50$ACP:myacp.EXE        {where myacp.EXE is the
                                 executable ACP image  }
$ EXIT                          {to terminate the process }
```

- ACP Process Name – must be specified, conventionally with the name of the ACP image. If allowed to default, a duplicate ACP name error will occur if any other installed ACP uses the LOGINOUT mapping.

- UIC – should be set to match a user account that has, as defaults, all the VMS privileges and quotas needed to run the ACP. (Otherwise, LOGINOUT will constrain the ACP to the privileges and quotas of the account that was used to start up CM50S, regardless of the privileges and quotas specified on the ACP installation screens.) When a UIC is specified for LOGINOUT, the ACP must be installed with the "NOUAF" status flag set. (Otherwise, the User Authorization File for the dispatching user will override the UIC requested in the process creation.)

Note that if a more complex command file is used as SYS$INPUT for an ACP, then its first command must invoke an image that calls CM50_SET_ACP (or ACPTRP) as its first executable statement and its next-to-last command (just before the $ EXIT) must invoke an image that calls PRGTRM as its final executable statement. No additional calls to either CM50_SET_ACP or PRGTRM may be made by any image invoked in the command file.

5.6 ACP INSTALLER ACTIVITY LOG SCREEN

When KP4/DISP LOG is selected from the ACP Operations screen, the Installer Activity Log screen is displayed. This screen displays the 1000 most recent changes that have been made to all ACPs in the system. The date and time of the change, ACP name, full pathname, installation mode, ACIDP name, CG Port Number, and the change that was made to the ACP are all shown. Up to eight entries can be displayed on the screen at one time.

```
INSTLOG                        INSTALLER ACTIVITY LOG              11 APR 91 14:09

Date              ACP Name     Action                              ACIDP          CG
11-APR-91 12:39 DDT30          Connected ACP                       C19_ALF         2
                    mode = normal
11-APR-91 12:36 DDT30          Disconnected ACP                    C19_ALF         2

11-APR-91 12:36 DDT30          Changed ACP Mode                    C19_ALF         2
                    from RESTRICTED to NORMAL
11-APR-91 12:36 DDT30          Changed ACP Mode                    C19_ALF         2
                    from unchanged  to RESTRICTED
11-APR-91 06:35 DDT30          Changed ACP Mode                    C19_ALF         2
                    from NORMAL     to RESTRICTED
11-APR-91 06:34 DDT30          Connected ACP                       C19_ALF         2
                    mode = normal
11-APR-91 06:24 DDT30          Disconnected ACP                    C19_ALF         2

28-MAR-91 08:54 RGBAD          Uninstalled ACP

PF4     PF2                                                  G8       KP8      KP2      G2
QUIT    HELP     JUMP TO:_______ (DDMMMYY)                   FIRST    PRIOR    NEXT     LAST
```

5.6.1 ACP Installer Activity Log Screen Fields

- Date—The date and time that the change was made to the ACP. The number to the left of the name corresponds to that entry's position in the Installer Activity Log.

- ACP Name—This is the name that the ACP was installed under. It is displayed directly underneath the date and time and may differ from the executable filename. This field is never blank.

- Action—This identifies the change that was recorded, using one of the following keywords:
  ACTIVATED—The ACP was activated interactively.
  CHANGED—The ACP's installation mode was changed. Both old and new mode values are shown.
  CONNECTED—The ACP was connected to an ACIDP. The mode (Test, Restricted, or Normal) is shown on the next line.

DEACTIVATED—The ACP was deactivated from the VAX. The next line shows whether the ACP was set to ABORT or to OFF.

INSTALLED—The ACP was installed. The next line shows the pathname of the executable image.

UNINSTALLED—The ACP was uninstalled.

- ACIDP—This is the name of the ACIDP that the ACP is connected to. This field may be blank if the ACIDP name is not relevant to the change that was made.

- CG—This is the CG Port Number that the ACP is associated with. This is an integer from one to four. This field may be blank if the CG number is not relevant to the change that was made.

5.6.2 ACP Installer Activity Log Screen Function Keys

PF4/QUIT—Cancels viewing of the Installer Activity Log and returns to the ACP Operations screen.

PF2/HELP—See heading 3.1.3.

JUMP TO:_____—Type the starting date in the format DDMMMYY (such as 20DEC88) and press <RETURN>. The first entry for that date is then displayed as the first entry on the screen. If an invalid date is entered, then an error message is displayed at the bottom of the screen. In either case, the screen then awaits user input. Pressing <RETURN> without entering a date causes an error message to be displayed at the bottom of the screen.

KP8/SCROLL UP—Scrolls to prior entries in the Installer Activity Log.

KP2/SCROLL DOWN—Scrolls to later entries in the Installer Activity Log.

5.7 TEST AND RESTRICTED MODES OF ACP OPERATION

The "Test" and "Restricted" modes of ACP operation help in the checkout of new or revised Advanced Control Programs by providing a controlled environment. Table 2-1 summarizes the ACP operating attributes for each operating mode.

Operation mode selection and change are made through the ACP Operations screen. For example, a program can be installed in "Test" mode. Then, after testing, its mode can be changed to "Restricted" or "Normal" mode.

5.7.1 Test Mode Operation

This is the most tightly controlled mode of operation. It uses preassigned test values for input and prevents the export of any output values to the LCN.

The test-input values are established when you build the Data Definition Tables to be referenced by the ACP. If a test value is not defined for a referenced input parameter, currently accessed data is stored as the input value.

ACPs in test mode are activated through the ACP Operations screen. While in test mode, an ACP also can be activated by operator demand at a Universal Station if the ACT_TYPE parameter value of its attached ACIDP includes demand.

5.7.2 Restricted Mode Operation

Restricted mode operation differs from test mode only in the use of "live" values from the LCN rather than test input values. Outputs to the LCN are blocked and program activation is the same as in test mode.

5.8 RECOVERY OF AN ACP FROM ABORT STATE

When an ACP has been aborted by its own action, it can be restored by operator action at a Universal Station (see the parameter OPER_DMD in the *Computer Gateway Parameter Reference Dictionary*). The ACIDP also can be reset from a VAX terminal by disconnecting and then reconnecting the ACP.

If the ACP was aborted by VMS and is still shown to be in RUN state by the Universal Station or by the ACP Status display, it can be forced to the ABORT or OFF state by deactivation through the ACP Operations screen.

DATA DEFINITION TABLES
Section 6

This section explains how you prepare, view, and modify the special tables that are used by some of the User Interface routines.

6.1 OVERVIEW OF DDT PREPARATION

The use of Data Definition Tables was discussed at heading 2.1.2. In this section you will find out how DDT tables are created and modified. Each DDT table begins with the preparation of a source file that contains all the information required by its type. Table 6-1 illustrates some characteristics of the five table types.

Table 6-1 — DDT Table Types and Data Types

DDT TABLE TYPES	ALLOWED DATA TYPES IN THE SAME DDT TABLE	PT COUNT MAX
Input	Real Integer **or** Time **or** Internal Entity ID ASCII **or** String **or** External Entity ID Enumeration **or** Ordinal	300 (see note following
Generic Input	Up to four of the following types: Real Integer Time Internal Entity ID ASCII String External Entity ID Enumeration **or** Ordinal	300 (see note following)
Output	Real Integer **or** Time **or** Internal Entity ID ASCII **or** String Enumeration **or** Ordinal	300 (see note following)
Generic Output	Up to four of the following types: Real Integer Time Internal Entity ID ASCII String Enumeration **or** Ordinal	300 (see note following)
History	Real & Integer (See paragraph 6.3.4)	24

NOTE

The number of values (Pt. Count Max) that can be transferred using a DDT is limited by internal buffering on the LCN. While each DDT can include up to four different data types, the total values for all types cannot exceed 300. The buffer can transfer a total of only 152 ASCII, String, and External Entity ID data types. For DDTs that contain a mix of ASCII and other data types, the exact maximum is difficult to compute but the total point count can be 300 if the number of the ASCII types plus enumerations is less than 91.

Note that the standard input and output DDTs can be used with either the normal DDT Get and DDT Store functions or with the Generic DDT calls. However, Generic input and output DDTs can be used only with the Generic DDT calls.

You begin creation (or modification) of a DDT with selection of the "Edit" function (KP1) on the DDT Operations screen after entering the table name and identifying its type. A DDT Data Entry screen suitable to the specified type appears and provides you with a "template" to be filled in. You continue with the Edit function until information on all the point.parameters to be accessed has been entered. The command that saves the source file brings you back to the DDT Operations screen for the "Build" step.

The Build command (G1) prepares the DDT in two stages. It first reads and verifies the source file, meanwhile creating several component files to hold information about this DDT. While a DDT is being built (or rebuilt), it cannot be accessed by any other process.

If the source file is error free, the Builder program proceeds to the second stage, the external-to-internal name conversion. Source-file verification can be done without the CG, but external-to-internal name conversion requires access to the LCN.

NOTE

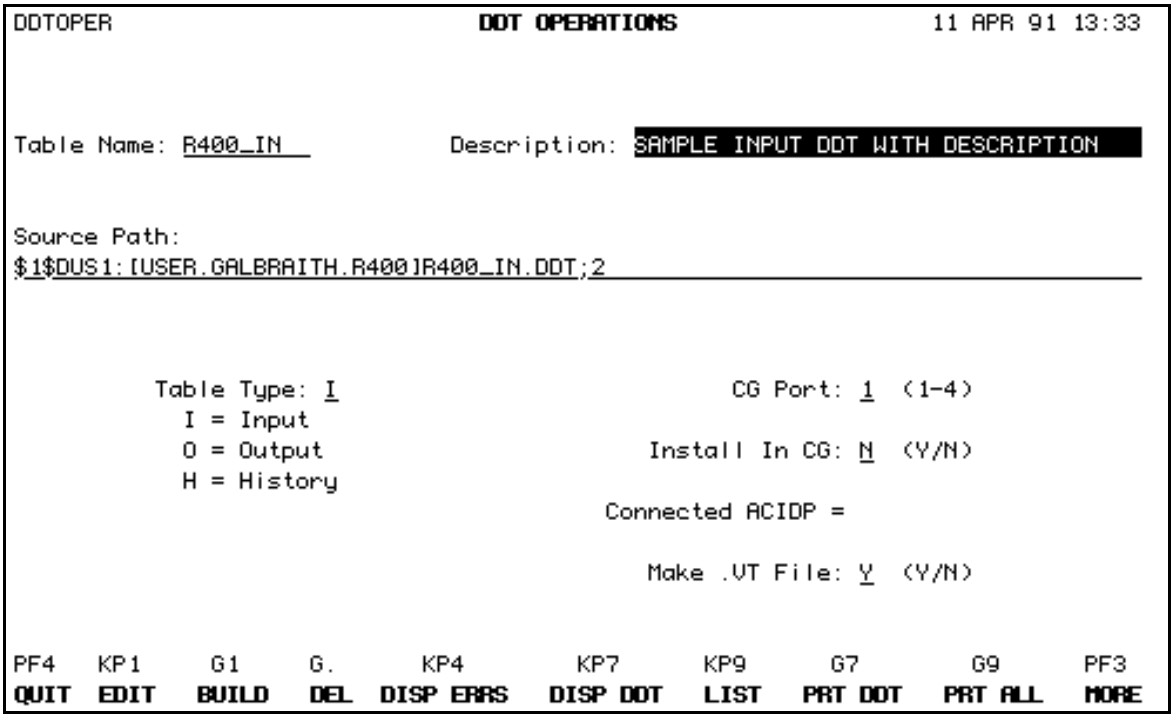
Data Definition Tables need to be rebuilt whenever the LCN database is changed in a significant manner, such as by the rebuild or deletion of data points referenced by that DDT.

As indicated above, the Data Definition Table is not a single file, but includes both the source file and a set of files that are created during the build process. Each component file of a DDT is identified by the table name and a suffix that denotes the file type and use.

Details of the DDT Operations screen and the hierarchy of subsidiary displays that it invokes begin at heading 6.2.

6.2 DDT OPERATIONS SCREEN

All Data Table operations such as `EDIT`, `BUILD`, `DELETE`, `LIST`, and `PRINT` begin at the DDT Operations screen (accessed from the CM50S Main Menu). At this screen you enter preliminary information before proceeding to other displays to complete the desired work.



The hierarchy of displays that start from the DDT Operations screen is

DDT Operations
- Edit DDT—KP1
- DDT Error Summary —KP4
- DDT Detail Description —KP7
- DDT List —KP9
- ACIDP-DDT Prefetch —G3

6.2.1 DDT Operations Screen Fields

- `Table Name`—Enter the one-to-nine-character name of a new DDT or the name of a DDT that has already been built. This field must be filled in for all cases except `Quit`, `Help`, `Print All`, and `List`. After the DDT name is entered, a user action routine checks to see if the DDT already exists. If the DDT exists, the rest of the screen is automatically filled in to show the existing parameters. Modifications can be made to any field on the screen.

- `Description`—This is a 36-character description of the DDT and is an optional entry. If the specified DDT exists, this field is filled in automatically. This field can either be filled in or left blank.

- `Source Path`—This is the full, up to 80-character pathname of the DDT source file. This field allows operations to be performed on DDTs that do not reside in the current working directory and also allows the DDT name to be different than the DDT source file name. If this field is left blank, it is assumed that the DDT source file resides in the current working directory and has the same DDT name. The ".DDT" suffix in the pathname is an automatic default. Note that the pathname is dynamically expanded to specify the volume and extension names, and that the 80-character maximum length applies to the expanded pathname.

  When an existing DDT is displayed, the source path is shown automatically and includes the version number of the source file used to build it. (The version number is shown as ";nn" at the end of the source path.) Note that if the source file has been changed since the last successful build, the initial display still shows the old version number. To call up the most recent version of the source file, either delete the version number from the screen, or replace it with version 0.

- `Table Type`—This field indicates whether the DDT is for Input, Output, or History. If the DDT already exists, this field is filled in automatically. This field is used only when creating or modifying a DDT. The possible values for table type are:

  I — DDT will be used for input from the LCN.
  i — Input DDT, but can only be used with the CM50_DDT_GETGEN function.
  O — DDT will be used for output to the LCN.
  0 — Output DDT, but can only be used with the CM50_DDT_STORGEN function.
  H — DDT will be used for input from the History Module on the LCN.

- `CG Port Number`—This field indicates the number of the CG port that is used for data requests. This must be a number from 1 to 4. The default is `1`. Entry in this field is required for only the `Build`, `Install in CG`, `Remove from CG`, and `Modify connection` functions.

- `Install in CG`—This field indicates whether or not the DDT is to be CG resident. The default is `N`. It is used only when performing the `Build` function on Input DDTs.

- `Connected ACIDP`—If the DDT is connected to an ACIDP, the ACIDP name appears in this field, otherwise the field is blank. This is a display-only field.

- `Make .VT File`—Normally set to “N” to maximize throughput. If set to “Y” (generally for debugging), the most recently transferred values are kept on disk so that they can be viewed using the DDT Detail Description displays (see heading 6.6).

  Note that use of a .VT file reduces performance and keeps the DDT locked until the values are recorded on disk. This blocks the concurrent use of that DDT by more than one process.

6.2.2 DDT Operations Screen Function Keys

The menu on the bottom of the screen displays the names of the function keys that can be pressed to perform DDT functions such as editing, building, printing, and deleting a DDT. Each function on this menu is described below.

`PF4/QUIT`—Exits DDT Operations and returns to the CM50S Main Menu.

KP1/EDIT—This operation allows for the creation and modification of DDT source files. The Table Name and Table Type fields must be filled in, otherwise an error message is displayed on the bottom of the screen. A data entry screen for the type specified by the Table Type field is displayed. Refer to heading 6.3 for more information on editing DDTs.

G1/BUILD—Verifies that all of the point.parameters referenced by the DDT exist on the LCN and then adds the DDT to the CM50S known list of DDTs. The Table Name, Table Type, CG Port Number, and Install in CG fields must be filled in, otherwise an error message is displayed at the bottom of the screen. If an error occurs while building the DDT, an error message appears on the bottom of the screen. If the DDT is built without errors, a "success" message appears at the bottom of the screen. In either case, the screen then awaits user input.

If you are rebuilding an existing DDT, the system deletes and replaces the existing set of DDT files automatically.

NOTE

Before invoking the Build command, you first need to consider three available options:

- If the DDT is to reside in the CG, you must change the value of "Install in CG:" to Y.
- If data prefetch is wished, you must use the ACIDP-DDT Prefetch display (heading 6.4) to select the ACIDP and the trigger types.
- If you want the most recently transferred values to be available on disk for viewing, you must specify that a .VT file is to be made.

If the source file has been modified, make certain that the source path shows the desired version number. A blank version number (no ";nn" at the end of the source path) causes the latest version of the source to be used for the build.

G./DEL—Removes the DDT from the CM50S list of known DDTs and deletes all of that DDT's binary files. The source file is not deleted. The Table Name field must be filled in, otherwise an error message is displayed on the bottom of the screen. If an error occurs while deleting the DDT's binary files, an error message appears at the bottom of the screen. If the DDT is deleted without errors, a "success" message appears at the bottom of the screen. In either case, the screen then awaits user input.

KP4/DISPLAY ERRORS—This operation displays the error file created during a DDT build. The Table Name field must be filled in with the name of an existing DDT. If the DDT does not exist, an error message is displayed at the bottom of the screen. The user can scroll up and down through the error file. Refer to heading 6.5 for more information on this display.

KP7/DISPLAY DDT—This operation displays the description of one DDT on the screen. The Table Name field must be filled in with the name of a previously built DDT. If the DDT does not exist, an error message is displayed at the bottom of the screen. The user can page forward and backward through the DDT definition or jump to specific Point.Parameters in the DDT. Refer to heading 6.6 for more detailed information on this function.

KP9/LIST—This operation displays all of the installed DDT names and other pertinent information on the screen. This includes each DDT's name, description, status, number of points, whether it is for input, output, or history, the CG number, and whether or not it is CG-resident. The data in all fields is ignored. The user can page forward and backward through the DDT names. The format is the same as for the DDT Detail Description displays (see heading 6.6).

G7/PRINT DDT—This operation prints the description of the DDT to the default printer. The Table Name field must be filled in with the name of an existing DDT. If the DDT does not exist, an error message is displayed at the bottom of the screen. After the DDT description has been sent to the printer, the message DDT description has been printed appears at the bottom of the screen.

G9/PRINT ALL—This operation prints all of the built DDT names and other pertinent information to the default printer. The report contains each DDT's name, description, status, number of points, whether it is for input, output, or history, the CG Port number, and whether or not it is CG-resident. The data in all fields is ignored. After the DDT Table has been sent to the printer, the message DDT List has been printed appears at the bottom of the screen. Refer to heading 6.6 for the screen equivalent of this function.

PF3/MORE—Displays the alternate function key menu (shown following) at the bottom of the screen.

PF4	PF2	G0	G,	G3	PF3
QUIT	HELP	INST IN CG	REM FROM CG	MOD CONNECTION	MORE

The additional functions on this menu are

PF2/HELP—See heading 3.1.3.

G0/INSTALL IN CG—Installs the DDT in the CG. The Table Name field must be filled in.

G,/REMOVE FROM CG—Removes the DDT from the CG. The Table Name field must be filled in. Note that on most personal computer keyboards this VT00 keypad comma key is labeled as "+".

G3/MOD CONNECTION—Causes the ACIDP - DDT PREFETCH screen to be displayed.

6.3 EDIT DDT SCREENS

When the KP1/EDIT key is pressed at the DDT Operations screen with all required fields filled in, a DDT Entry template is displayed. The type of template (Input, Output, or History) to appear depends on the type of DDT specified. While in this edit mode, all field entries are checked for syntax errors before accepting data from the screen. Fields identified as having errors will blink. All identified errors must be corrected before any function other than QUIT (PF4) or DELETE (G.) will be executed.

NOTE

When a new Input or Output DDT is specified, a blank template for data type Real is the first template displayed. If the Input or Output DDT already exists, the first template appears with field information filled in. Its data type depends on what type(s) have previously been entered. A screen function key (KP,) controls change of I/O template data type.

LCN tags can be entered in any order, but the DDT Source file generated by this utility is sorted so that all tags of the same data type are grouped together. A maximum of four different data types can be included in a DDT. Any Reals will come first, followed by Integers, 24-character ASCII, Enumerations, Ordinals, Internal Point IDs, External Point IDs, Times, and 40-character ASCII (in that order).

Entry templates exist for

Input-Real	Output-Real
Input-Integer	Output-Integer
Input-ASCII	Output-ASCII
Input-Enumerated	Output-Enumerated
Input-Ordinal	Output-Ordinal
Input-Time	Output-Time
Input-String	Output-String
Input-Internal_ID	Output-Internal_ID
Input-External_ID	
History	

The table type and data type appear in the upper left-hand corner of the screen. Also shown at the top of the screen are the table name and an entry number (within the data type). History DDT screens do not have entry numbers.

The same function key menu is used by all Input and Output template displays; the History template display uses a different function key menu.

6.3.1 DDT Table Entry Rules

The paragraphs that follow explain the general requirements for DDT Source Table data entry. Then, starting at paragraph 6.3.2, the specific data-entry requirements for each individual data point-type are discussed. (See the *Hiway Gateway Parameter Reference Dictionary, Application Module Parameter Reference Dictionary,* and *Computer Gateway Parameter Reference Dictionary* for valid parameter names and their data types.)

NOTE

For values of Enumeration type there is a choice between character-string and ordinal-value presentations. You can include one type or the other, but not both, in an Input or Output DDT.

Output of enumeration strings is limited to the set of standard enumerations as defined in the *Parameter Reference Dictionaries*. Parameters of type self-defining enumeration accept only ordinal values of the enumerations.

6.3.1.1 General Information

The following rules apply to data entry for all DDT templates:

All data is entered left-justified (no leading blanks). Trailing blanks are ignored.

Real numbers between 0 and 1 do not require a zero to precede the decimal point (e.g., 0.05 and .05 are both acceptable). Whole real numbers need not contain a decimal point (e.g., 10, 10., and 10.0 are all acceptable).

There is no need to segregate points whose data is held in differing nodes on the same LCN into separate DDTs. On the contrary, the combining of points with different data owners into a single DDT makes better use of the LCN's distributed processing.

6.3.1.2 Data-Specific Information

POINT NAME and PARAMETER—Both point name and parameter must be entered for all template types. For input tables, the point name and parameter specify where the input value is to be fetched from. For output tables, the point name and parameter specify where the value is to be sent.

Note that several template types use additional point and parameter name entries—for Bad Value Substitution and for use in algorithms 6 through 9.

The point name must be from one-to-16 characters long and begin with "A..Z," "0..9," or "$"(for system data entities). Characters within the point name must be "A..Z," "0..9" or "_" (underscore). Consecutive underscore characters within a name are not permitted. Any trailing underscores are ignored.

The point name can be preceded by a pinid (1 or 2 characters followed by a backslash "\" delimiter) if the LCN supports Network Gateway routing. The name may not include embedded spaces. A space or null character (hex 00) is treated as a terminator, so any following characters will be lost.

The parameter must be from one to eight characters long and begin with "A..Z." Characters within the parameter must be "A..Z," "0..9," or "_" (underscore). Any trailing underscore characters are ignored.

Parameters can be subscripted (to allow accessing single elements of parameter arrays) by enclosing the subscript value in parentheses immediately following the parameter name; for example, name(nn). A maximum of 14 character positions is allowed for a subscripted parameter name.

DESTINATION—The destination field specifies where in the application program's data array for that data type the value is to be stored. (Used only if you do not wish values to be stored in locations corresponding one-to-one with positioning of entries in the DDT.) The destination value is an offset from the base location for that data type's data array. Thus, if the input-array table name is INTVAL, a destination of 23 results in that value being stored in array-location INTVAL(23).

If no destination is entered (either blanks or underscores), the value is placed into the program's array in a location corresponding to its position in the DDT for that data type. For example, if the DDT is comprised of 15 data points of type Real followed by 25 data points of type Integer and no destination is entered for any of the points, the values are stored as follows: The real values are stored in the real-array REAVAL(1) through REAVAL(15), and the integer values are stored in the integer-array INTVAL(1) through INTVAL(25). (REAVAL and INTVAL represent the names for the real array and integer array used in the program call to the interface routine.)

If the destination is specified as 0, the value is not returned to the calling program in its data array. This feature may be useful when an LCN parameter is included in a DDT solely for use in a bad value substitution or algorithm calculation. If a Value Table is saved for the DDT, the value for this point is included in the DDT Detail displays.

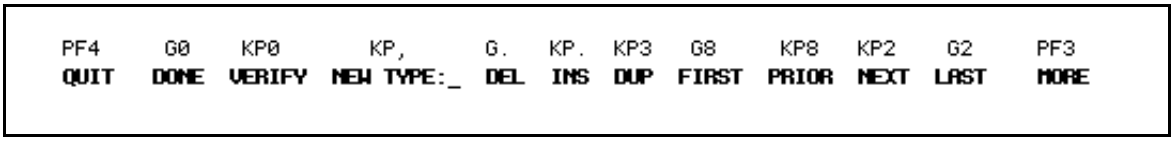
CAUTION

If a DESTINATION or SOURCE location is specified, the user is responsible for avoiding duplicate assignments. The default assigns the sequential entry number as the destination or source without checking for conflicts. It is recommended that you either leave the destination/source blank for **every** entry in a DDT, or that you assign an explicit destination/source to **every** entry.

SOURCE—The source parameter specifies from where in the program's data array the value is to be taken. As with "destination," this value identifies which element of that data type's value array is to be stored in this tag. A source value of 0 (zero) is invalid.

6.3.1.3 Input/Output DDT Data Entry Screen Function Keys

Following is an illustration of the first set of common Function Keys that appear with any of the Input or output DDT data entry displays.



Each function on this primary input DDT menu is described below.

PF4/QUIT—Cancels data entry and returns to the DDT Operations screen. If any changes have been made, you will be asked to verify that you wish to quit without saving changes.

G0/DONE—Sorts the defined points by data type then writes the file to the Source Pathname. Control then returns to the DDT Operations screen.

KP0/VERIFY—Makes a single-point call to see if the point.parameter exists on the LCN. The call is made to the CG port that was specified on the DDT Operations screen. A message appears at the bottom of the screen stating whether or not the point.parameter exists on the LCN. This allows you to create DDT source files without the CG being connected, then verify them later.

KP,/NEW TYPE—Allows you to change the type of data being entered. The cursor is placed at the entry field at the bottom of the screen, and the prompt: "Real, Integer, ASCII, Enum, Ord, Ptid, Xtern, Time or Stri & <RETURN>" appears on the bottom line of the screen. Type the first letter of the new type and press <RETURN>. An entry screen for that data type then appears on the screen, ready for input. Pressing PF4 cancels the operation. Note that there is no restriction on the order of data entry; the system orders the filled in templates by data type at the end of the entry session. Within each data type, the filled in templates are maintained in order of entry.

G./DELETE—Deletes the current entry being displayed on the screen. You are prompted to confirm the deletion. The entry that follows the deleted entry is then displayed. If the current entry is the last one of the current data type, a blank entry screen is displayed. If the Deletion request is cancelled (by answering NO to the confirmation prompt), the screen is restored to show the previous values for that entry.

KP./INSERT—Inserts a blank entry of the current data type BEFORE the current entry.

KP3/DUPLICATE—Adds a duplicate of the current entry directly AFTER the current entry. This is useful when multiple entries with the same format are desired.

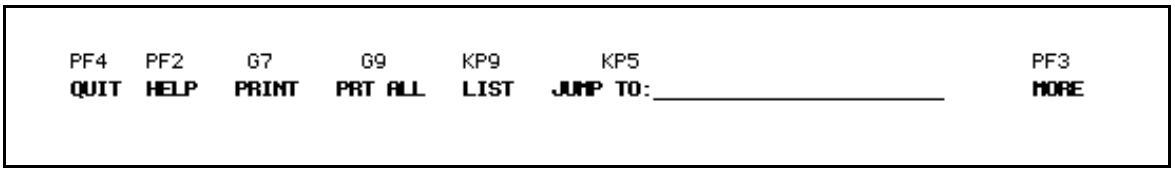
G8/FIRST—Displays the first entry of the current data type.

KP8/PRIOR—Displays the previous entry of the DDT. If the current entry is the first one, an error message is displayed at the bottom of the screen.

KP2/NEXT—Displays the next entry of the current data type. If the current entry is the last one, a blank entry screen is displayed.

G2/LAST—Displays the last entry of the DDT.

PF3/MORE—Displays the alternate function key menu (shown following) at the bottom of the screen.



The additional functions on the alternate Input DDT menu are described below.

PF2/HELP—See heading 3.1.3

G7/PRINT—Prints the current entry to the default printer.

G9/PRINT ALL—Prints a summary of all entries in the DDT to the default printer. The Point Name, Parameter, Source/Destination, Data Type, is shown for each entry. Refer to heading 6.3.2.4 for the screen equivalent of this function.

KP9/LIST—Displays all of the entries on the screen in the same format as the PRINT ALL function. Refer to Section 6.3.2.4 for more detailed information on this function.

KP5/JUMP TO—Allows you to move from existing entry template to entry template within the DDT. The cursor is positioned at the first JUMP TO field at the bottom of the screen. If the function key menu containing the JUMP TO field is not visible on the screen, it will be displayed before the cursor is positioned. Type the entry number, point.parameter, or .parameter and press <RETURN>. If the point.parameter is not found or if the entry number is not valid, an error message is displayed at the bottom of the screen. To cancel the operation without jumping to a new entry, press PF4.

6.3.1.4 Data Type Compatibilities

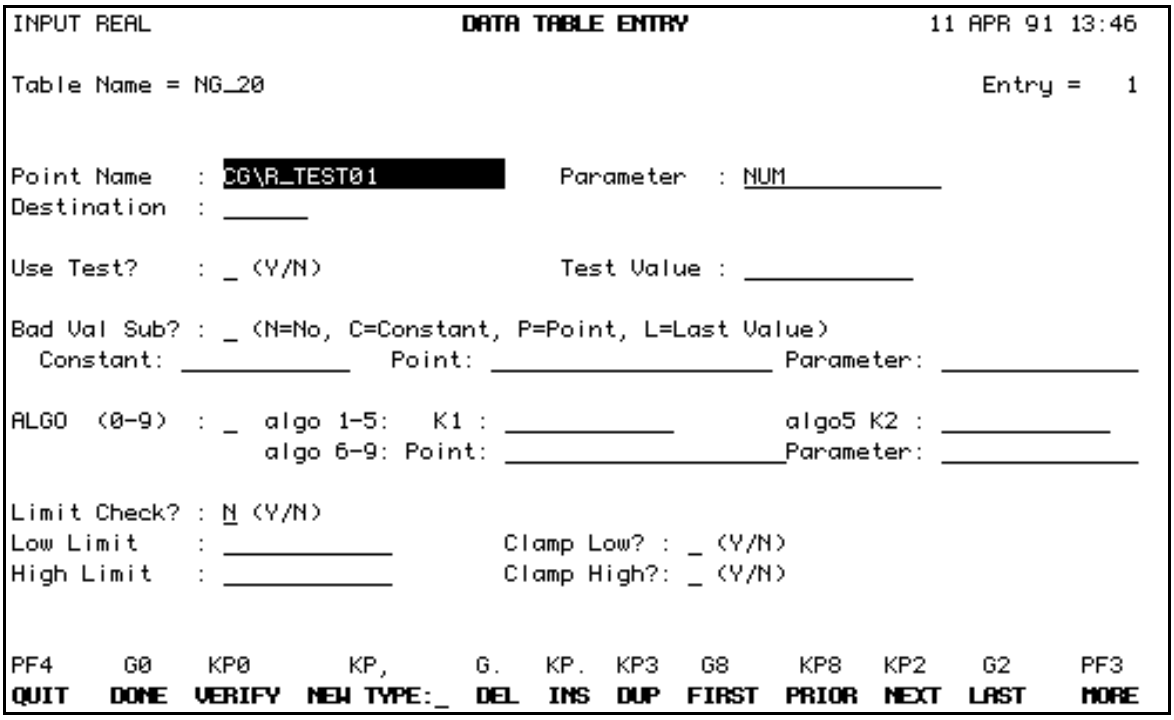
The DDT edit operation will not allow the creation of a DDT that violates the Data type requirements for the DDT interface. Therefore, the source DDT may never include more than four different data types. Enumerations and Ordinal values may not be mixed within the same DDT. Entity IDs may be stored only in internal (Ptid) format, so the Xtern data type is permitted only in Input DDTs.

No distinction is made between standard and generic DDTs during the edit operation. However, if a DDT includes more than one member of either of the following lists of data types, it will be flagged as Generic when the DDT is built:

1) Integer, Ptid (internal entity id), and Time;
2) ASCII, String, and Xtern (external entity id).

6.3.2 Input DDT Data Entry Screens

6.3.2.1 Input-Real Data Entry Screen



The only required entries are `Point Name` and `Parameter`.

`Point Name, Parameter,` and `Destination`—As described at heading 6.3.1.2.

`Use Test?`—Specifies whether or not a test value is to be used when the ACP's Installation Mode is "Test." If a test value is to be used, change to `Y` and enter the value in `Test Value`. Otherwise, `Test Value` remains blank.

`Bad Val Sub?`—Specifies whether or not to perform a Bad Value Substitution, and source of the substitution value.

If the default value of `N` remains, the `Constant`, `Point Name`, and `Parameter` fields remain blank.

If `C` is entered, the `Constant` field must be filled in.

If `P` is entered, both the `Point Name` and `Parameter` fields must be filled in. The substitution Point Parameter named must be defined in a previous `Point Name` in this DDT and be of the same data type as this point. Note: The substituted point.parameter value is the input value, not its calculated-result value. Scaling is performed as specified by `ALGO` for this data point.

If L is entered, Constant, Point Name, and Parameter remain blank. This bad value replacement option retains the algorithm's most recently calculated output value. Note that this substitution can only occur if a values table (.VT) is maintained, for this DDT and that a good value has been returned at least once.

ALGO—The choice of scaling algorithm defines the type of calculation to be performed on the input value. The legal algorithm entries are

0 – No action—K1 and K2 are ignored.

1 – V + K1
2 – V - K1
3 – V * K1
4 – V / K1

Enter K1.

(When algorithm 4 is selected, K1 cannot be 0.0)

5 – V * K1 + K2—Enter both K1 and K2.

6 – V + VI
7 – V - VI
8 – V * VI
9 – V / VI

Enter both Point Name and Parameter.

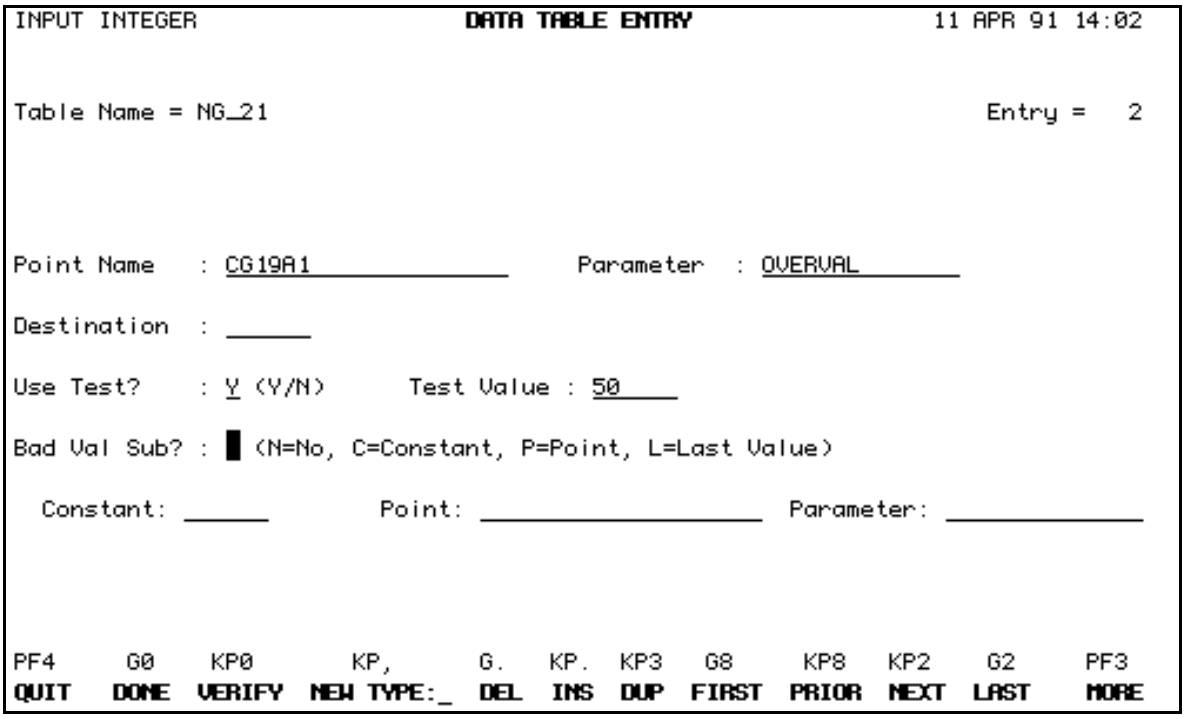
Where:

V = this point/parameter's input value.
K1 and K2 = real-number constants, either positive or negative.
VI = the input value of a point/parameter previously defined in this table and of the same data type as this point/parameter.

Limit Check—Allows for the returned value (after any value substitution and value calculation) to be checked against limiting values. If Y is entered, one or both of Low Limit and High Limit must be filled in depending on whether one limit or both are to be checked. Limit-check values are real-number constants, either positive or negative. When either limit is exceeded, and the corresponding clamp option is not selected, the value is changed to NaN.

Clamp Low?/Clamp High?—When clamping is selected, if the corresponding limit value is exceeded, the value is clamped (set) at the limit. If clamping is not selected and the limit is exceeded, the value is replaced by the bad value constant (-0).

6.3.2.2 Input-Integer Data Entry Screen



The only required entries are `Point Name` and `Parameter`.

`Point Name, Parameter,` and `Destination`—As described at heading 6.3.1.2.

`Use Test?`—Specifies whether or not a test value is to be used when the ACP's Installation Mode is "Test." If a test value is to be used, change to `Y` and enter the value in `Test Value`. Otherwise, `Test Value` remains blank.

`Bad Val Sub?`—Specifies whether or not to perform a Bad Value Substitution, and source of the substitution value.

If the default value of `N` remains, the `Constant`, `Point Name`, and `Parameter` fields remain blank.

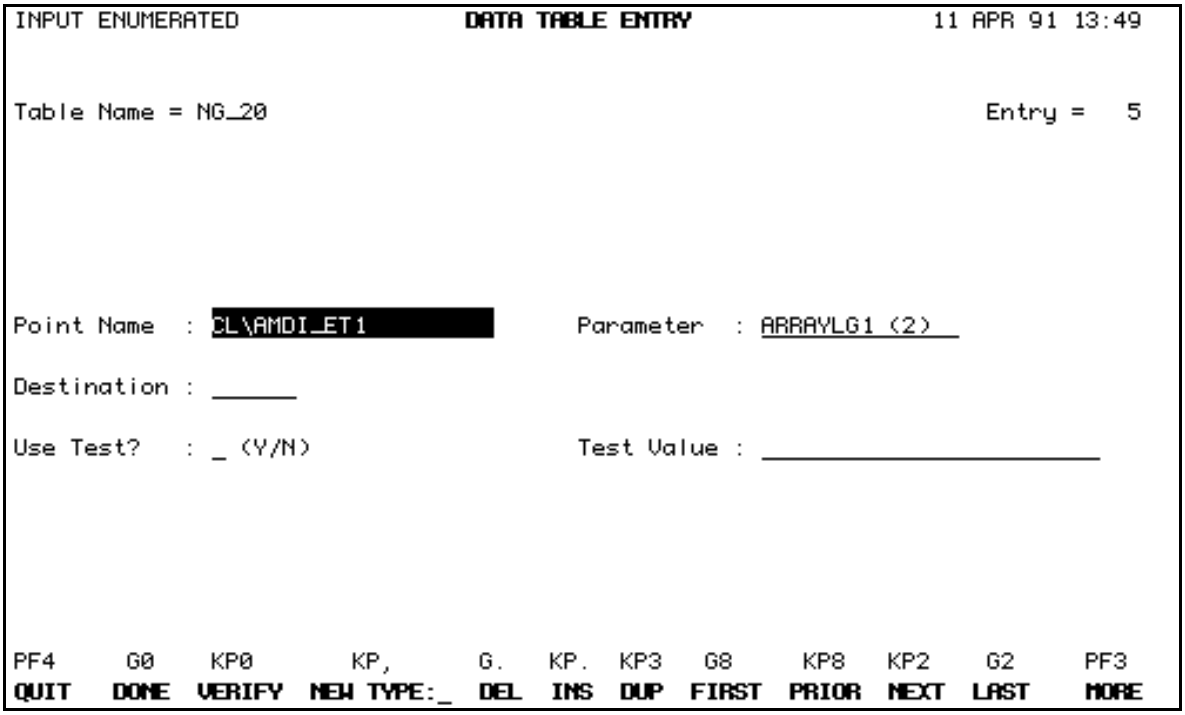
If `C` is entered, the `Constant` field must be filled in.

If `P` is entered, both the `Point Name` and `Parameter` fields must be filled in. The substitution Point. Parameter named must be defined in a previous `Point Name` in this DDT and be of the same data type as this point. Note: The substituted point.parameter value is the input value, not its calculated-result value. Scaling is performed as specified by `ALGO` for this data point.

If `L` is entered, `Constant`, `Point Name`, and `Parameter` remain blank. The actual substitution cannot be made until at least one good value for the point has been obtained.

6.3.2.3 Other Input Data Entry Screens

The template as shown below is for ASCII values. The screens for the remaining data types are the same except for the data-type identifier in the upper left-hand corner which identifies the value type with the appropriate keyword: `ASCII, ENUMERATED, ORDINAL, INTERN. ID, EXTERN. ID, TIME or STRING.`



The only required entries are `Point Name` and `Parameter.`

`Point Name, Parameter,` and `Destination`—As described at heading 6.3.1.2.

`Use Test?`—Specifies whether or not a test value is to be used when the ACP's Installation Mode is "Test." If a test value is to be used, change to `Y` and enter the value in `Test Value`. Otherwise, `Test Value` remains blank. The format of the Test Value depends on the data type:

ASCII -- up to 24 ASCII characters
ENUMERATED -- up to 8 ASCII characters
ORDINAL -- Integer value of a defined Enumeration
INTERN. ID -- 4 Integer values separated by commas (e.g., 256,01,-234,0)
EXTERN. ID -- 16-character point name, optionally followed by a 2-character pinid (in positions 17 & 18)
TIME -- Integer number of seconds (The .1 millisecond tick count will be zero)
STRING -- up to 24 ASCII characters

Note that you can include only one or the other of the data types ENUMERATED or ORDINAL in an Input DDT.

6.3.2.4 Input DDT Point Summary Screen

This screen is reached by pressing KP9/LIST from any of the DDT entry screens while editing an Input DDT. The Table Name and Table Type appear at the top of the screen. Within a data type, the entries are listed in the order they were entered. Up to 12 entries can be displayed on the screen at one time.

```
PTSUMM                          DDT POINT SUMMARY                    11 APR 91 13:48

Table Name = NG_20                    Table Type = INPUT              Entries =  10

  ENTRY  POINT            PARAMETER            DATA TYPE          DEST.

     1   CG\R_TEST01      NUM                  REAL                  1
     2   CL\AMDI_ET1      ARRAYNM1 (1)         REAL                  2
     3   CL\AMDI_ET1      ARRAYNM1 (2)         REAL                  3
     4   R_TEST01         NUMARR (2)           REAL                  4
     5   CL\AMDI_ET1      ARRAYLG1 (2)         ENUMERATED            1
     6   CL\AMDI_ET1      ARRAYEN1 (2)         ENUMERATED            2

     7   R_TEST01         CENM                 ENUMERATED            3
     8   CG\R_TEST01      SENM                 ENUMERATED            4
     9   CL\AMDI_ET1      ARRAYTM1 (1)         TIME                  1
    10   CL\AMDI_ET1      ARRAYTM1 (5)         TIME                  2




PF4      PF2       G9        Enter                G8       KP8      KP2      G2
QUIT     HELP      PRINT     RETURN               FIRST    Page UP  DOWN     LAST
```

Each column of the Point Summary screen is described below.

- Entry—The entry number of the point.parameter. This is equivalent to the index of the STATUS TABLE that is described in the *CM50S User Manual.*
- Point—Point names as specified on the DDT entry screens.
- Parameter—Parameter names as specified on the DDT entry screens.
- Type—This is the Data Type of the value to be retrieved or stored to the LCN. This column always contains one of the following keywords: REAL, INTEGER, ASCII, ENUMERATED, ORDINAL, INTERN. ID, EXTERN. ID, TIME, or STRING.
- Dest(ination)—For an Input DDT, this column shows the array element in which to store the value being retrieved from the LCN. The array would be the REAL_VALUES_ARRAY, INTEGER_VALUES_ARRAY, etc. as described in Section 3. If applicable, this column shows the user-specified destination; otherwise the calculated relative destination is displayed.

The following function keys are used to manipulate this display.

PF4/QUIT—Cancels viewing of the entry list and returns to the DDT Operations screen. If any changes were made while editing, the user is asked to verify that they want to quit without saving changes.

PF2/HELP—Displays a short help message on the bottom of the screen. If more help is needed, press PF2 again and an entire screen of help is displayed. Pressing <RETURN> from this help screen re-displays the entry screen.

G9/PRINT—Prints the entire list of entries to the default printer.

ENT/RETURN—Returns to the Data Entry screen that was being displayed when the LIST function was requested.

G8/FIRST—Displays the first page of entries.

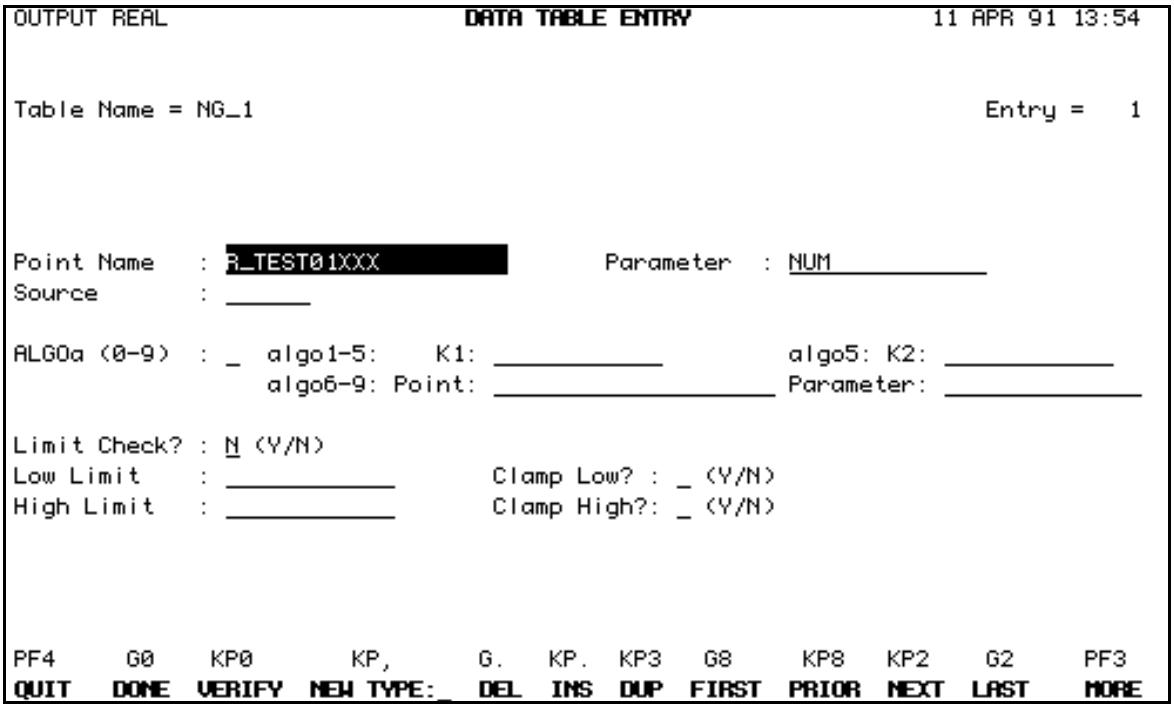
KP8/PRIOR—Displays the previous page of entries. If the first page of entries is already on display, an error message is displayed.

KP2/NEXT—Displays the next page of entries. If the last page of entries is already on display, the first page of entries appears.

G2/LAST—Displays the last page of entries.

6.3.3 Output DDT Data Entry Screens

6.3.3.1 Output-Real Data Entry Screen



The only required entries are `Point Name` and `Parameter`.

`Point Name, Parameter,` and `Source`—As described at heading 6.3.1.2.

ALGO—The choice of scaling algorithm defines the type of calculation to be performed on the point.parameter value. The legal algorithm values are

0 – No action—K1 and K2 are ignored.

1 – V + K1
2 – V - K1
3 – V * K1
4 – V / K1

Enter K1.

(When algorithm 4 is selected, K1 cannot be 0.0)

5 – V * K1 + K2—Enter both K1 and K2.

6 – V + VI
7 – V - VI
8 – V * VI
9 – V / VI

Enter both Point Name and Parameter.

Where:

V = this point/parameter's value as stored in the output-data array.
K1 and K2 = real-number constants, either positive or negative.
VI = the output data-array value of a point/parameter previously defined in this table and of the same data type as this point/parameter.

`Limit Check`—Allows for the value to be stored (after any algorithmic calculation) to be checked against limiting values. If `Y` is entered, one or both of `Low Limit` and `High Limit` must be filled in depending on whether one limit or both must be checked. Limit-check values are real-number constants, either positive or negative. When either limit is exceeded, and the corresponding clamp option is not selected, the value is changed to NaN.

`Clamp Low?/Clamp High?`—When clamping is selected, if the corresponding limit is exceeded, the value is clamped (set) at the limit.

6.3.3.2 Other Output Data Entry Screens

The template as shown below is for Enumeration values. The screens for the remaining data types are the same except for the data-type identifier in the upper left-hand corner which identifies the value type with the appropriate keyword: `ASCII, ENUMERATED, ORDINAL, INTERN. ID, TIME or STRING.`

```
OUTPUT STRING                    DATA TABLE ENTRY                  11 APR 91 13:55


Table Name = NG_1                                                    Entry =   7









Point Name : CG19A3                Parameter : STR40_____

Source     : _____









PF4     G0     KP0       KP,       G.    KP.   KP3    G8      KP8    KP2     G2      PF3
QUIT    DONE   VERIFY    NEW TYPE:_  DEL   INS   DUP    FIRST   PRIOR  NEXT    LAST    MORE
```

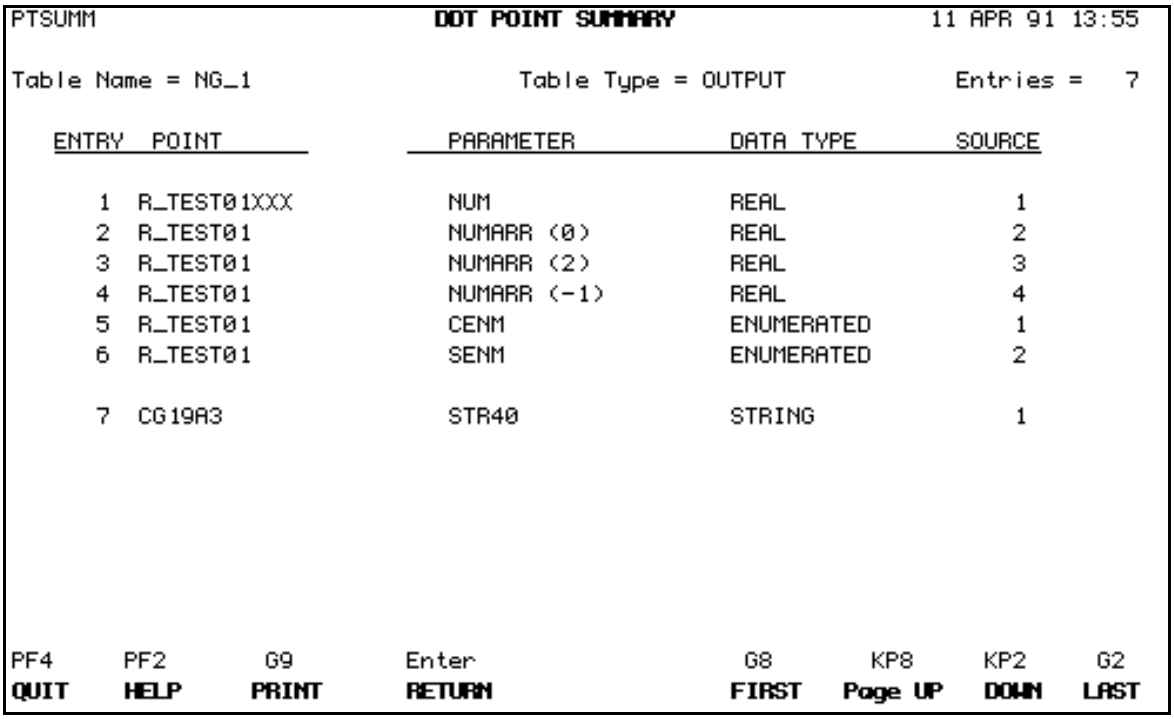
The only required entries are `Point Name` and `Parameter`.

`Point Name, Parameter,` and `Source`—As described at heading 6.3.1.2.

Note that you can include only one, or the other, of the data types ENUMER or ORDINAL in an Output DDT.

6.3.3.3 Output DDT Point Summary Screen

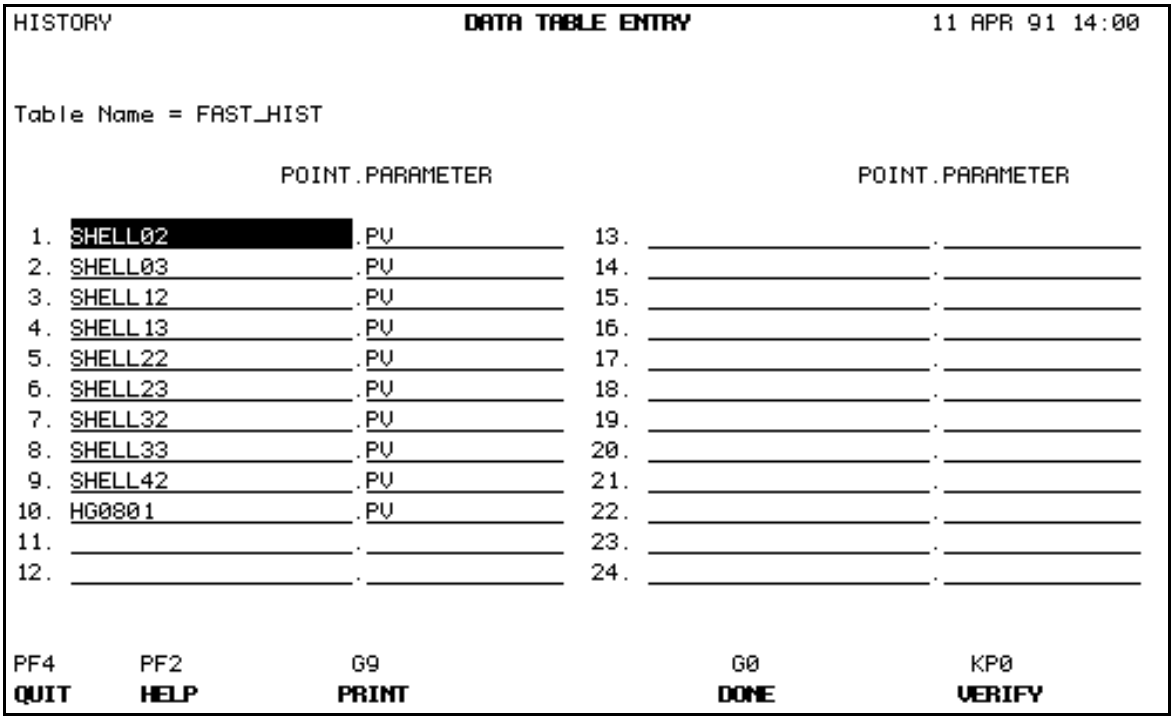
This screen is reached by pressing KP9/LIST from any of the DDT entry screens while editing an Output DDT. This screen is the same as the one for the Input DDT except that the Dest column has been changed to Src.



The information on this screen is the same as the Point Summary screen for Input DDTs described at heading 6.3.2.4, except for the Source column described below.

- Src—This column shows the array element number that contains the value to be stored to the LCN. The array would the REAL_VALUES_ARRAY, INTEGER_VALUES_ARRAY, etc. as described in Section 3. If applicable, this column shows the user-specified source; otherwise, the calculated relative source is displayed.

6.3.4 History DDT Data Entry Screen



6.3.4.1 History DDT Data Entry Fields

`Point` and `Parameter`—Enter the names of up to 24 point.parameters as specified in 6.3.1.2. This is the only information needed for a History DDT.

Note that continuous history averages consist only of Real values. Snapshots can be either Real values or ordinal enumerations of digitals. Because of message-size limitations, the maximum number of point.parameters in a request for one hour of snapshots is 19.

6.3.4.2 History DDT Function Key Menu

`PF4/QUIT`—Cancels data entry and returns to the DDT Operations screen. If any changes were made, you are asked to verify that you want to quit without saving changes.

`PF2/HELP`—See heading 2.8.3

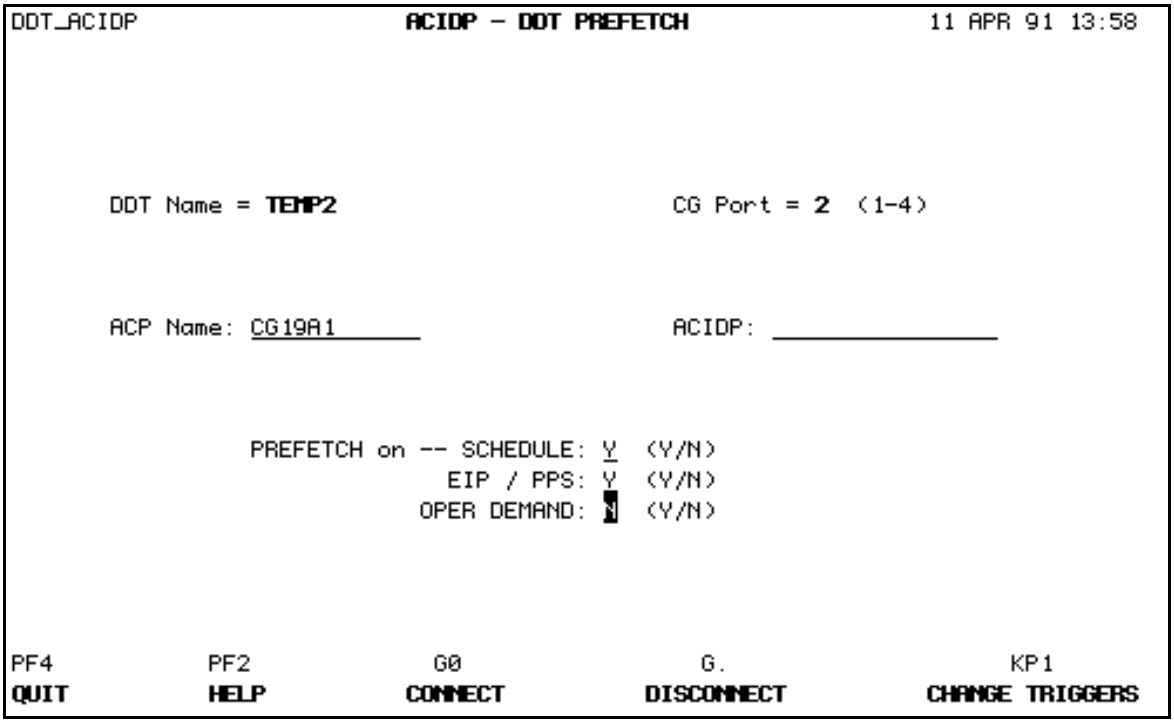
`G9/PRINT`—Prints all entries in the DDT to the default printer.

`G0/DONE`—Saves the data into a .DDT file and returns to the DDT Operations screen.

`KP0/VERIFY`—Makes single point calls to the CG port specified on the DDT Operations screen to see if the point.parameters exist on the LCN. Because there can be up to 24 point.parameters, this can take some time. If some point.parameters are not found on the LCN, the erroneous entries blink and an appropriate error message appears at the bottom of the screen. If all point.parameters exist on the LCN, a message appears on the bottom of the screen to signal completion of this function.

6.4 ACIDP-DDT PREFETCH SCREEN

This screen is displayed as the result of the G3/MOD CONNECTION selection from the DDT Operation Screen. From this display you can modify the ACIDP-DDT connection and/or Triggers by making the appropriate entries on the screen then invoking the desired operation from the function key menu at the bottom of the screen.



You enable data prefetch for the points specified by an input DDT by making it CG-resident, connecting it to an ACIDP, and selecting one or more event "triggers." Data prefetch means that the LCN data requests are made in parallel with the ACP turn-on request, thus reducing the lag time between ACP activation and availability of LCN data to it. The events that can trigger an ACIDP are Schedule (cyclic or periodic), Point Process Special (PPS), and operator demand.

The unique functions on this screen are

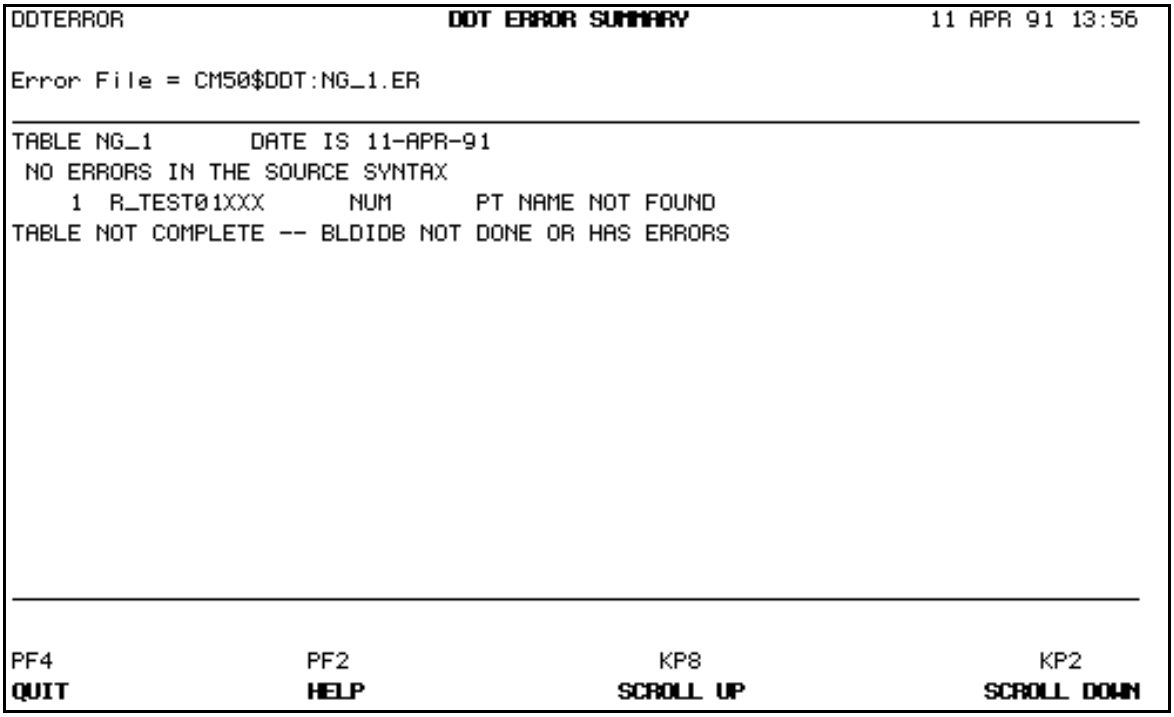
G0/CONNECT—Connects the specified ACIDP and DDT.

G,/DISCONNECT—Disconnects the specified ACIDP and DDT.

KP1/CHANGE TRIGGERS—Establishes/changes the Triggers based on the entries in the "Prefetch on" fields.

6.5 DDT ERROR SUMMARY DISPLAY

The DDT Error Summary screen is displayed as the result of KP4/Display Errors selection from the DDT Operations Screen and shows the error file created during a DDT installation.



The following function keys are used to manipulate this display.

PF4/QUIT—Cancels viewing of the Installer Activity Log and returns to the ACP Operations screen.

PF2/HELP—See heading 3.1.3.

KP8/SCROLL UP—Scrolls to prior entries in the DDT Error Summary.

KP2/SCROLL DOWN—Scrolls to later entries in the DDT Error Summary.

6.6 DDT DETAIL DESCRIPTION DISPLAYS

When KP7/DISP DDT is selected at the DDT Operations screen, the detailed description of a DDT can be viewed. The table name, table type, description, build status, CG Port Number, ACIDP name, and source path are all displayed at the top of the screen. A CG Port Number with an asterisk (*) next to it means that the DDT is CG-resident.

The full description of each entry in the DDT is shown, including the test value, bad value substitution, algorithm, low and high limits, value in, value out, and source/destination. Input, Output, and History DDTs have slightly different formats; therefore, a sample of each screen is shown.

The DDT Display screens list the entries of the DDT in a specific order. All of the Real entries are shown first, followed by Integer, ASCII, and Enumerated (or Ordinal) entries in that order. If the DDT does not contain any Real entries, the display starts with Integers; if there are no Integers, the display starts with ASCII entries, and so on.

6.6.1 Function Keys for DDT Detail Displays

All of the Detail Description Displays use the same set of Function Key attributes.

PF4/QUIT—Cancels viewing of the DDT Display and returns to the DDT Operations screen.

PF2/HELP—See heading 3.1.3.

KP5/JUMP TO:—Type the "point_name," "point.parameter," ".parameter," or the number of the point to be displayed and press <RETURN>. That point is then displayed as the first entry on the screen. The valid entry numbers appear to the left of the point names. If an invalid entry is made, an error message is displayed on the bottom of the screen. In either case, the screen then awaits user input. Pressing <RETURN> without entering a point number causes an error message to be displayed at the bottom of the screen.

G9/PRINT—Prints the entire DDT Display to the default printer.

G8/FIRST—Displays the first page of the DDT's Display. If the first page of entries is already on display, a message stating so appears at the bottom of the screen.

KP8/PRIOR—Displays the previous page of the DDT Display. If the first page of entries is already on display, a message stating so appears at the bottom of the screen.

KP2/NEXT—Displays the next page of the DDT Display. If the last page of DDT entries is already on display, a message stating so appears at the bottom of the screen.

G2/LAST—Displays the last page of the DDT Display. If the last page of DDT entries is already on display, a message stating so appears at the bottom of the screen.

6.6.2 Data Presentation Rules for DDT Detail Displays

One of the options of a DDT is the maintenance of a Values Table (.VT) that holds the values (before and after any transformations) from the last use of the DDT. The values in this table are included in the Detail Displays. The "In value" is the raw value received by the DDT call. The "Out values" reflect any changes made by Bad Value substitutions and algorithm calculations. The form and amount of data presented by xxx is determined by the "Build Status" of the table to be examined/changed.

When Table Build Status is `Complete` and the `Make .VT File` option was used — All information is presented. If the table has not yet been used by an application program, or no .VT file was made, the date and time field in the heading is blank, as are the value fields for each item and the ACP ST field. Once the table has been used by an application program, all fields are shown. Note: If Table Processing is suppressed on a DDT call, that call will not update the .VT file, so the displayed values would reflect the most recent time that Table Processing was invoked.

Any bad values are displayed as `????-NNN-???`, where NNN is a code that indicates why the value is considered to be bad. See Appendix A for a listing of these codes.

Good values are displayed as required by data type.

Real—If the absolute value is greater than 999999 or less than 0.0001, it is shown in exponential form as `n.nnnnnE+ee`; otherwise, the value is displayed as `nnnnnn.nnnn`. A leading sign is shown only if the value is negative.

Integer—Shown as 1-to-5 digits, with a leading sign only if the value is negative.

ASCII—Shown as a character string of 1-to-24 characters.

Enumerations—Shown as a character string of 1-to-8 characters.

Ordinals—Shown as a positive integer of 1-to-5 digits.

Internal Id—Shown as 4 integer values, separated by commas.

External Id—Shown as a character string of 1 to 18 characters (point name in positions 1-16, and pinid in positions 17-18).

Time—Shown as a positive integer of Seconds followed by a comma and a positive integer count of Ticks.

String—Shown as a character string of 1-to-24 characters; the contents of characters 25 through 40 are not saved in the .VT file.

When Table Build Status is `Incomplete` or `Complete-Errors`—The source file is good, but the internal-point IDs and the parameter table are not complete. The table cannot be used by an application program. The date and time field and ACP ST field are shown as blanks. The values-in and values-out fields are displayed as blanks. All other information is shown.

When Table Build Status is `Source Errs`—The source file is not valid and the transform file is not complete. Only the table name, table description, table purpose, and table-build status are shown.

6.6.3 DDT Display Screens for Input Tables

The `Value In` and `Value Out` fields of these displays show the actual data values that are passed between the VAX and the LCN if Table Processing is used.

- `Value In`—This is the value that was retrieved from the LCN. It is shown only if the DDT build status is `COMPLETE` and if the DDT has previously been used to access LCN data by an application program.

- `Value Out`—This is the processed value that is passed to the application program after a bad value substitution or an algorithm has been performed on it. It is shown only if the DDT build status is `COMPLETE` and if the DDT has previously been used to access LCN data. If the entry has a `DESTINATION` of 0 (zero), then the value is followed by two asterisks (**), meaning that the value was not stored in the application program's array.

- `Destination`—This is the array element in which to store the value being retrieved from the LCN. The values would be stored in the `REAL_VALUES_ARRAY`, `INTEGER_VALUES_ARRAY`, etc., as described in Section 3.

6.6.3.1 Input-Real DDT Display

```
DDTDISP                              DDT DISPLAY                        17 APR 91 16:01

DDT Name: IN_SAMPLE              Description= FOR EXAMPLE DDT PROGRAMS
Purpose=  INPUT      Status= COMPLETE           CG=2     ACIDP=
Source Path =
$1$DUS1:[USER.GALBRAITH.NGTESTS.DDT]IN_SAMPLE.DDT;8
-------------------------------------------------------------------------------
  1 Point =    SHELL02          Param = PV              Type = REAL
    Use Tst = No
    BV Sub  = No
    Algo    = 0 (None)
    Lim Ck  = No
    Val In  =                 Val Out =                 Dest  =   1

  2 Point = CG\SHELL03          Param = PV              Type = REAL
    Use Tst = No
    BV Sub  = No
    Algo    = 0 (None)
    Lim Ck  = No
    Val In  =                 Val Out =                 Dest  =   2
-------------------------------------------------------------------------------
PF4     G9      JUMP TO (point.param or #):          G8     KP8    KP2     G2
QUIT   PRINT    _____________________________      FIRST  PRIOR  NEXT    LAST
```

6.6.3.2 Input-Integer DDT Display

```
DDTDISP                              DDT DISPLAY                    22 MAY 91 15:28

DDT Name: I001I                  Description=
Purpose=  INPUT      Status= COMPLETE           CG=1     ACIDP=
Source Path =
$1$DUS1:[CM50S.R4_TEST.DDT]I001I.DDT;4
-------------------------------------------------------------------------------

  1 Point =    HG0801           Param = OVERVAL          Type = INTEGER
    Use Tst = No
    BV Sub  = No
    Val In  = 1               Val Out = 1                Dest  =   1









-------------------------------------------------------------------------------
PF4     G9      JUMP TO (point.param or #):            G8     KP8    KP2     G2
QUIT   PRINT    _____________________________________  FIRST  PRIOR  NEXT   LAST
```

6.6.3.3 Input-Other DDT Point Types

```
DDTDISP                              DDT DISPLAY                    17 APR 91 16:01

DDT Name: IN_SAMPLE              Description= FOR EXAMPLE DDT PROGRAMS
Purpose=  INPUT      Status= COMPLETE           CG=2     ACIDP=
Source Path =
$1$DUS1:[USER.GALBRAITH.NGTESTS.DDT]IN_SAMPLE.DDT;8
-------------------------------------------------------------------------------

  5 Point =    C19_DICK         Param = PTDESC           Type = ASCII
    Use Tst = No
    Val In  =                                            Dest  =   1
    Val Out =









-------------------------------------------------------------------------------
PF4     G9      JUMP TO (point.param or #):            G8     KP8    KP2     G2
QUIT   PRINT    _____________________________________  FIRST  PRIOR  NEXT   LAST
```

6.6.4 DDT Display Screens for Output Tables

The `Value In` and `Value Out` fields for these displays show the actual data values that are passed between the VAX and the LCN if Table Processing is used.

- `Value In`—This is the value that was retrieved from the ACP's array. It is shown only if the DDT build status is `COMPLETE`, and if the DDT has previously been used by an ACP to access the LCN.
- `Value Out`—This is the processed value that is stored to the LCN after a bad value substitution or an algorithm has been performed on it. It is shown only if the DDT build status is `COMPLETE`, and if the DDT has previously been used by an ACP to access the LCN. If the entry has been marked `No Store` by the ACP, the value is followed by two asterisks (**), meaning that the value was not stored to the LCN.
- `Source`—This is the array element that holds the value to be stored to the LCN. The values would be found in the REAL_VALUES_ARRAY, INTEGER_VALUES_ ARRAY, etc., as described in Section 3.

6.6.4.1 Output-Real DDT Display

```
DDTDISP                              DDT DISPLAY                    17 APR 91 16:04

DDT Name: OT_SAMPLE              Description=
Purpose=  OUTPUT     Status= COMPLETE          CG=2     ACIDP=
Source Path =
$1$DUS1:[USER.GALBRAITH.NGTESTS.DDT]OT_SAMPLE.DDT;3
-------------------------------------------------------------------------------
  1 Point =    SHELL02          Param = PV              Type = REAL
    Algo    = 0 (None)
    Lim Ck  = No
    Val In  =                 Val Out =                 Src   =   1

  2 Point =    SHELL03          Param = PV              Type = REAL
    Algo    = 0 (None)
    Lim Ck  = No
    Val In  =                 Val Out =                 Src   =   2

  3 Point = CG\SHELL04          Param = PV              Type = REAL
    Algo    = 0 (None)
    Lim Ck  = No
-------------------------------------------------------------------------------
PF4     G9      JUMP TO (point.param or #):          G8     KP8    KP2     G2
QUIT   PRINT    _____________________________       FIRST  PRIOR  NEXT    LAST
```

6.6.4.2 Output-Integer DDT Display

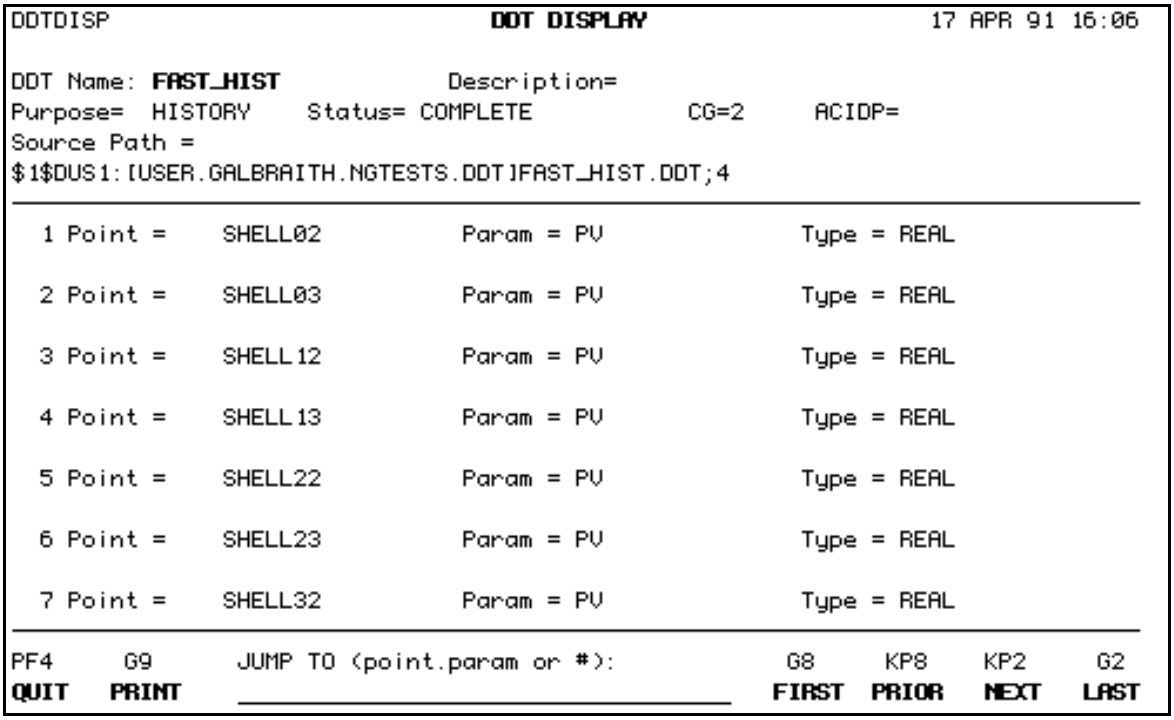
```
DDTDISP                              DDT DISPLAY                    17 APR 91 16:08

DDT Name: OUT_INTEG              Description= ADJUST OVERVAL SETTINGS
Purpose=  OUTPUT     Status= COMPLETE            CG=1     ACIDP=
Source Path =
DISK$PUBLIC_1:[USER.GALBRAITH]OUT_INTEG.DDT;1
-------------------------------------------------------------------------------
  1 Point =    HG0801            Param = OVERVAL          Type = INTEGER
    Val In  =                   Val Out =                  Src   = Src   =

  2 Point = CG\HG0802            Param = OVERVAL          Type = INTEGER
    Val In  =                   Val Out =                  Src   = Src   =

  3 Point = CG\HG0803            Param = OVERVAL          Type = INTEGER
    Val In  =                   Val Out =                  Src   = Src   =

  4 Point = CG\HG0804            Param = OVERVAL          Type = INTEGER
    Val In  =                   Val Out =                  Src   = Src   =


-------------------------------------------------------------------------------
PF4     G9      JUMP TO (point.param or #):          G8     KP8    KP2     G2
QUIT    PRINT   _____________________________        FIRST  PRIOR  NEXT    LAST
```

6.6.4.3 Output-Other DDT Point Types

```
DDTDISP                              DDT DISPLAY                    17 APR 91 16:04

DDT Name: OT_SAMPLE              Description=
Purpose=  OUTPUT     Status= COMPLETE            CG=2     ACIDP=
Source Path =
$1$DUS1:[USER.GALBRAITH.NGTESTS.DDT]OT_SAMPLE.DDT;3
-------------------------------------------------------------------------------
  5 Point =    C19_DICK          Param = PTDESC           Type = ASCII
    Val In  =                                             Src   =   1
    Val Out =









-------------------------------------------------------------------------------
PF4     G9      JUMP TO (point.param or #):          G8     KP8    KP2     G2
QUIT    PRINT   _____________________________        FIRST  PRIOR  NEXT    LAST
```

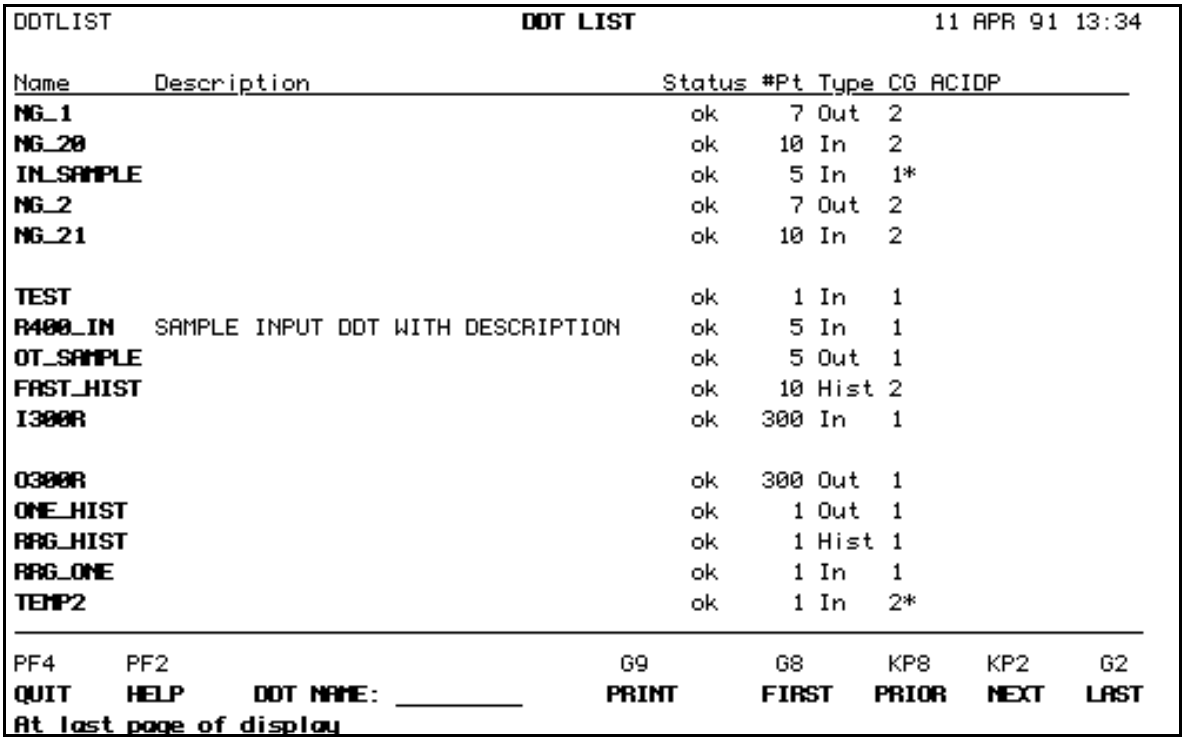
6.6.5 DDT Display Screen for History Tables

The History DDT Display screen lists the entries of the DDT in the same order as the Point Summary screen. Up to seven points can be displayed on the screen at one time.



6.7 DDT LIST DISPLAY

When KP9/LIST is chosen from the DDT Operations screen, the DDT List screen is displayed. This screen displays the name of every built DDT, its description, status, number of points, type, CG Port Number and whether or not the DDT is CG resident. Up to 15 entries can be displayed on the screen at one time.



6.7.1 DDT List Screen Data Fields

- DDT Name—The name of a DDT that has already been built.
- Description—This is the up to 36 character description that was associated with the DDT when it was built. If no description was entered upon building the DDT, this field is blank.
- Status—This is the completion status of the DDT. It contains one of the following keywords:

  Comp—The only status where the table is available for ACP use.

  Src Err—The table build could not be complete because of source file errors.

  Incomp—No source file errors, but the table could not be built because of errors between the VAX and the CG. See the error file for the reason.

  Errors–One or more point or parameter names are in error. See the error file for the reason.

- # Points—This is the number of points in the DDT.

- Type—This is the type of the DDT and contains one of the following keywords: In for an Input DDT, G.In for a Generic-only Input DDT, Out for an Output DDT, G.Out for a Generic-only Output DDT, or Hist for a History DDT.

- CG—This is the CG Port Number that the DDT is associated with. An asterisk (*) displayed to the right of the CG Port Number means that the DDT is CG resident.

6.7.2 DDT List Screen Function Keys

PF4/QUIT—Cancels viewing of the DDT Summary and returns to the DDT Operations screen.

PF2/HELP—See heading 3.1.3

DDT NAME:—Type the name of the DDT to be displayed and press <RETURN>. That DDT is displayed as the first entry on the screen. If an invalid DDT Name is entered, an error message is displayed at the bottom of the screen. In either case, the screen then awaits user input. Pressing <RETURN> without entering a DDT name causes an error message to be displayed at the bottom of the screen.

G9/PRINT—Prints the entire list of DDTs to the default printer.

G8/FIRST—Displays the first page of DDTs If the first page of entries is already on display, a message stating so appears at the bottom of the screen.

KP8/PRIOR—Displays the previous page of DDT entries. If the first page of entries is already on display, a message stating so appears at the bottom of the screen.

KP2/NEXT—Displays the next page of DDT entries. If the last page of entries is already on display, a message stating so is displayed at the bottom of the screen.

G2/LAST—Displays the last page of DDTs. If the last page of entries is already on display, a message stating so is displayed at the bottom of the screen.

6.8 DDT SOURCE FILE PREPARATION RULES

DDT source files can be created programmatically as text files with a maximum per-record length of 80 characters. The points must be grouped by data type and ENUMERation and ORDINAL points cannot be requested in the same DDT.

Each DDT Source file is composed of point definitions grouped by data type and separated by data type identifier records. Each point definition is composed of a point identifier record which can be followed by optional records to specify processing option values. Blank records can be embedded anywhere within the file to make it easier for humans to read. A DDT Source file containing three different data types would be organized thus:

Table Purpose & Data Type record
Description record (optional)
Point Definition
Point Definition
etc.
New Data Type record
Point Definition
Point Definition
etc.
New Data Type record
Point Definition
etc.

Each Point Definition is composed of a Point Name record followed by zero or more optional records that define how the point's data is to be processed. Note that when an optional record is omitted, a specific default value is used in the point's processing.

Note also that the DDT source files are data location sensitive and require fixed locations within the record for specified data items.

6.8.1 Data Type Records

The first record in a DDT source file is a special data type identifier record that also defines the table purpose (input, output, or history). The first record's format is:

```
purpos TABLE, DATA TYPE =type
```

Where "purpos" (in positions 1-6) is one of the following strings:

```
INPUTΔ    (where "Δ" represents a required space)
OUTPUT
HISTOR
```

And "type" (in positions 26-31) specifies the data type of the first set of points and must equal one of the following strings:

```
REAL
INTEGE
ASCII
ENUMER
ORDINA
INTERN
EXTERN
TIME
STRING
```

The only allowed data types in a History DDT are REAL, INTEGE, and ORDINA.

If the DDT includes more than one data type, then the point definitions for each new data type must be introduced by a record with the following format:

```
***  NEW DATA TYPE = type
```

Where positions 1 through 22 must contain the string Δ***ΔΔNEWΔDATAΔTYPEΔ=Δ ("Δ" represents a required space).

And "type" (in positions 23-28) specifies the data type of the next set of points as one of the following

```
REAL
INTEGE
ASCII
ENUMER
ORDINA
INTERN
EXTERN
TIME
STRING
```

6.8.2 Point Identification Record

The point definitions vary according to data type and table purpose and can contain optional records. Each DDT point definition begins with a point identification record formatted as follows:

```
POINT NAME =pt_name   PARAMETER=par_nm(n)       DESTINATION=
```

or

```
POINT NAME =longer_pt_name      par_nm(n)            SOURCE=
```

Where positions 1-12 must contain the string: POINTΔNAME =
Positions 13-32 contain the desired LCN Point name terminated by a space.
Positions 33-46 contain the LCN parameter name
Positions 61-66 can contain an optional ASCII representation of an integer to be used as an index into the value array (see SOURCE and DESTINATION field descriptions at heading 6.3.1.2). The word SOURCE or the word DESTINATION is required to specify which option is desired.

6.8.3 Point Option Records

The point identification record is followed by zero or more option records—again depending on data type and purpose of the DDT.

6.8.3.1 Test Option Record

For **INPUT** DDTs only, the record immediately following the point identification record can specify a value to be used with the test option:

```
USE TEST? =N   (Y/N)      TEST VALUE=
```

Where positions 1-12 must contain the string ΔUSEΔTEST?
Position 13 is Y or N (if N, the following value is ignored)
Positions 34- contain the test value (following spaces are optional). The maximum value length varies by type: 12 characters for numbers (no imbedded spaces)
8 characters for enumerations
24 characters for other ASCII (embedded spaces ok)
The default (if this record is omitted) is the same as an "N" record—no test value.

6.8.3.2 Input Integer Option Records

The following optional record pair is allowed with INPUT INTEGER point definitions:

```
BV SUBST? =
 Constant =                          pt_name              par_nm(n)
```

The upper record must begin with the string ΔBVΔSUBST?Δ= in positions 1-12. The character in position 13 determines the type of bad value substitution to be made: (N or a blank = no substitution, C = substitution of a constant, P = substitution of a value from another point.parameter, L = substitute the last good value from this point). The default (if this record pair is omitted) is the same as an "N" value—no bad value substitution.

The lower record is used only when the BV SUBST option of C or P is specified and must begin with the string ΔΔConstantΔ= in positions 1 through 12.
If the substitution value is C then:
positions 13-24 must contain the constant to be substituted (an embedded space terminates the field)
If the substitution value is P then:
positions 35-54 must contain the name of the alternate LCN point, terminated by a space, and
positions 59-72 must contain the name (and subscript if it is an element of an array) of the alternate parameter.
The alternate point and parameter must be defined in a previous Point Definition Record.

6.8.3.3 Input Real Option Records

The following optional records are allowed with INPUT REAL point definitions:

```
BV SUBST? =
 Constant =                          pt_name            par_nm(n)
    ALGO   n
LIMIT CK? =
LOW LIMIT =
HI LIMIT  =
```

The bad value substitution record pair have the same format used by input integers.

The Algorithm definition record specifies which algorithmic filter is to be applied to the data (the default is algorithm 0—no filter). This record must have the string ALGO in positions 6-9.
Position 13 contains the algorithm number (0 through 9). See heading 6.2.3.1 for a description of the INPUT REAL algorithms.
For Algorithms 1-5—positions 34-45 contain the constant value for K1.
For Algorithm 5—positions 55-66 contain the constant value for K2.
For Algorithms 6-9— positions 35-54 contain the related LCN point name and positions 59-72 contain the related LCN parameter name.

The Limit Check definition record specifies whether or not limit checking is to be done. This record must have the string ΔLIMITΔCK? in positions 1-10. Position 13 should contain "Y" for yes; "N" or a space indicates no. If this record is omitted, the default is no. When limit checking is set to yes and the value falls outside the limits, the returned status table will indicate an error.

The Low Limit definition record specifies whether or not limit checking applies to low values. This record must have the string ΔLOWΔLIMIT in positions 1-10.

Positions 13-24 should contain the constant to use as the low limit. If left blank, no low limit check is made (default).
Position 35 should contain "Y" if the value is to be clamped to to be not less than its low limit; "N" or space indicate no low limit clamp is to be set.

The High Limit definition record specifies whether or not limit checking applies to high values. This record must have the string ΔHIΔLIMITΔ in positions 1-10.

Positions 13-24 should contain the constant to use as the high limit. If left blank, no high limit check is made (default).
Position 35 should contain "Y" if the value is to be clamped to be not greater than its high limit; "N" or space indicate no high limit clamp is to be set.

6.8.3.4 Output Real Option Records

The following optional records are allowed with OUTPUT REAL point definitions:

```
     ALGO   n
LIMIT CK? =
LOW LIMIT =
HI LIMIT  =
```

The Algorithm definition record and the High/Low limit checks and clamp records for OUTPUT REAL follow the same descriptions used for INPUT REAL above.

6.8.4 Description Record

The following optional record is allowed only as the **second** record in the source file (immediately following the "purpose" record):

DESCRIPTION=text description for the DDT

Where position 1-12 contain the string:DESCRIPTION=
Positions 13-48 contain the desired descriptive text.

Note: If the DDT source file contains a Description record, that description will override any description when the DDT is built or rebuilt. If the DDT source is edited from the DDT Operations screen, the description displayed on the screen becomes part of the stored source file.

6.8.5 DDT File Data Entry Rules

Conversion of ASCII values into numbers follows the standard Pascal/FORTRAN conventions. Also, preceding blanks are permitted (to a maximum of 8); permitted characters are + - 0 1 2 3 4 5 6 7 8 9 . E (with . and E permitted only for real values; any other character is a terminator.

The use of preceding or embedded blanks in names is not permitted. The first blank character in a name field terminates that field.

UTILITY OPERATIONS
Section 7

This section introduces you to use of these CM50S Utility functions: the Task Scheduler, the MAKEINC utility, and LCN file transfer operations.

7.1 TASK SCHEDULER

In most environments a number of reports and routine operations need to be invoked on a regularly scheduled basis. The CM50S Task Scheduler runs these scheduled tasks in the background without operator intervention. While the Task Scheduler was designed for CM50S reporting, it is flexible enough to be used with any VAX/VMS command file or executable image.

The scheduler uses a **Command Table** of tasks to be scheduled. The user makes one entry in this table for each task that is to be run. Task Scheduler monitors the Command Table and automatically invokes each task whose scheduling parameters are met.

Task Scheduler is very similar to the UNIX scheduler function CRON. Familiarity with one implies familiarity with the other.

7.1.1 Task Scheduler Operation

The Task Scheduler invokes listed tasks at a scheduled time. On each cycle, it processes commands read from the user-created Command Table and sequentially invokes each of the tasks whose schedule matches the current time, date, and day-of-week criteria.

The scheduler is invoked on an "as needed" basis; CM50S does not require that the scheduler be used. The scheduler runs as a detached process and is initiated from the Task Scheduler menu (see heading 7.5). It can be stopped at any time using the "Stop Scheduler" selection option from the same menu. It is invoked for system-wide use under the process name CM50_SCHEDULER. This is the name you will see when doing a SHOW SYSTEM command. Once invoked, the scheduler remains active until the operator stops the scheduler, using the menu stop option.

Scheduled tasks can include executable images and DCL command files. Any required logical names must be in the System Logical Names table. Note that the console is the default SYS$INPUT and SYS$OUTPUT for all tasks run by the scheduler. These defaults can be redirected within .COM files to user-specified files. Programs that use terminal forms for I/O should not be run under the scheduler. Most CM50S terminal displays have an alternate form that can be run under the scheduler to produce printed output.

NOTE

Since the CM50_SCHEDULER is shared by multiple users:
1) All path names should be given in full (not assuming any home directory).
2) Jobs executed by the CM50_SCHEDULER cannot require any special VMS privileges or quotas.

7.1.2 Scheduler Command Table File

The Command Table file is named CM50_SCHED.TBL, and resides in a directory named CM50$CONTROL. You define and maintain it through the system editor. It is read initially when the scheduler is first invoked and is reread at 2-minute refresh intervals.

The Command Table file is free form, but each field must be separated by one or more spaces and there must be six fields for each line entry. It is recommended that the fields be maintained as aligned columns with a fixed number of spaces separating the fields. This is not a scheduler requirement. It is only needed to improve readability.

Figure 7-1 illustrates some example schedules that are explained following.

```
# MINUTE        HOUR      DAY-OF-MONTH   MONTH    DAY-OF-WEEK   COMMAND

0               17        J1,J32         *        *             @DIA0:[user]NEW_MONTH
0               8         *              *        1-5           @DIA0:[user]DAILY_REPORT
0,15,30,45      17-23     *              1-3      6             RUN DIA0:[user]FIRSTQ
0               0         1              *        0             PURGE CM50$DDT
*               *         *              *        *             @DIA0:[user]MINUTES
#  This is a comment line.
```

Figure 7-1 — Command Table Example

7.1.2.1 Command Line Explanations

Line 1. Invoke the command file named DIA0:[user]NEW_MONTH on January 1, and on February 1 at 5:00 P.M.

Line 2. Invoke the command file named DIA0:[user]DAILY_REPORT Monday through Friday at 8:00 A.M.

Line 3. Invoke the command file named DIA0:[user]FIRSTQ every Saturday during the months of January through March. The image is invoked at 15-minute intervals from 5:00 P.M. to 11:45 P.M.

Line 4. Execute the DCL command PURGE CM50$DDT on the first of each month, only if the first falls on a Sunday (0 = Sunday). The image is invoked at midnight.

Line 5. Invoke the command file @DIA0:[user]MINUTES every minute of every hour, of every day, of every week, of every month.

Line 6. The scheduler allows comments to be inserted in the Command Table file. All commented lines must begin with a pound sign (#).

7.1.2.2 Conventions Used in the Command Table

The only format requirements for the file are that each line entry must contain all six fields and each field must be separated by one or more spaces.

Each entry in the table must appear on a separate line of the file. The line is limited to 80 characters in length. The exact minute, hour, day, month, and day-of-week fields (values), dictate when the corresponding command field is invoked. The day-of-month field value can be represented in Gregorian or Julian form. (If you use the Julian form, be careful of leap years.)

There are several ways to specify the values for the first five fields of each line entry. A single numerical value, a range of values, or a list of values can be specified. An asterisk (*) placed in a field, specifies "at all times" during that interval.

Any line beginning with the number sign (#) is ignored by the scheduler and can be used for comments.

Day Of Week Values are: 0=Sunday . . 6=Saturday

7.1.3 Command Table Modification

The Command Table can be modified at any time. The scheduler rereads the table each 2-minute interval period. This updates (refreshes) the scheduling parameters and the scheduler re-schedules all tasks accordingly. This allows you to change the schedules without stopping the scheduler.

The system editor is invoked from the Task Scheduler menu (see heading 7.6). It automatically loads the Command Table for creation or modification. The values shown in Figure 7-2 represent all possible values for each field.

# MINUTE	HOUR	DAY-OF-MONTH	MONTH	DAY-OF-WEEK	COMMAND
0-59 *	0-23 *	1-31 * J1-J366	1-12 *	0-6 *	@(filename) or executable image name

Figure 7-2 — Command Table Fields

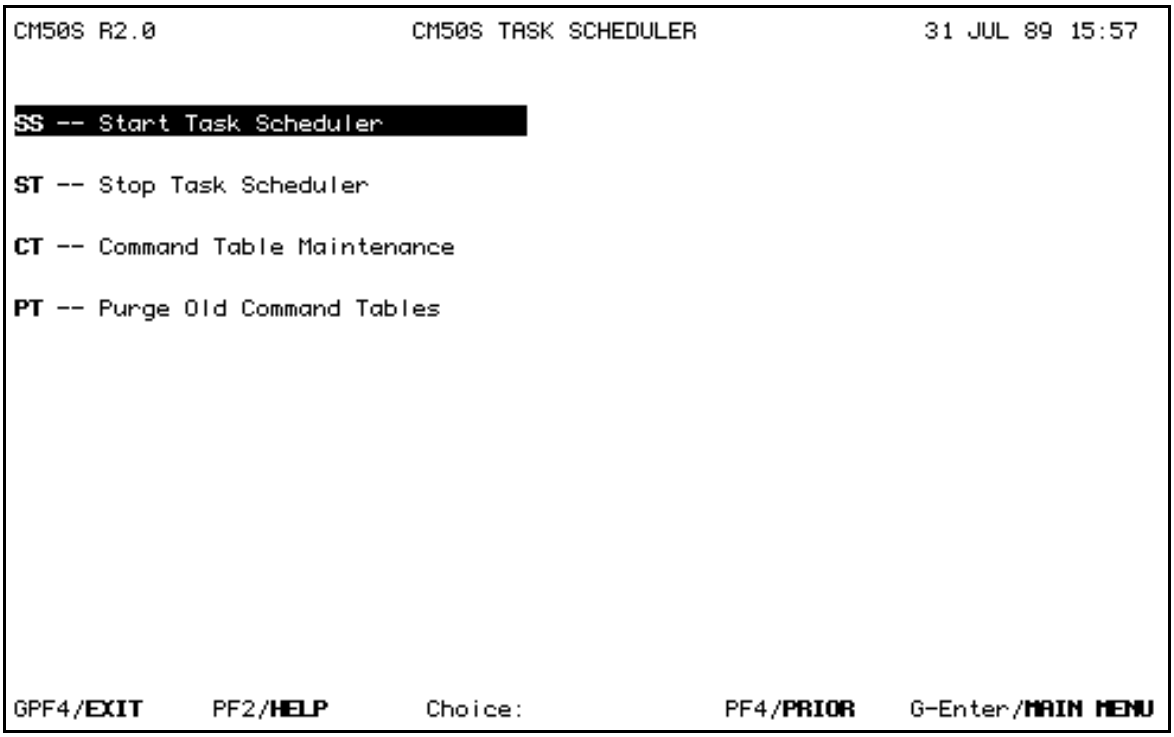
Remember, the file is free form, but one or more spaces must separate each field and there must be six fields. The table is more easily understood if the fields are maintained in columnar order, with sufficient distance apart for readability.

To invoke the system editor, select the Task Scheduler menu and key the characters CT (Command Table Maintenance), or use the cursor bar selection for CT. The system editor begins execution and automatically loads the Command Table file. (If the Editor comes up with a single display and an asterisk (*) prompt, type "chan<Enter>" to change it to full screen editing mode. When you are finished with the edit, return to the edit command line (control Z) and type EXIT. This saves the edited file. If you do not want to save the edits, type QUIT. In either case, you will be returned to the Task Scheduler menu.

7.1.4 Task Scheduler Menu

All operator interaction with the scheduler is through the Task Scheduler Menu. This menu is a submenu of, and is selected from, the CM50S Operations Main Menu.

The Task Scheduler menu appears on the screen as follows:



The top line of the menu displays the screen ID CM50S R2.0 (left-hand corner), the menu title (center), and the current date and time (right-hand corner).

The next four lines are the selection options. The selection can be made by keying the corresponding 2-character code into the choice field. A selection can also be made by using the reverse video cursor bar that highlights the selection option and then pressing return for the selection.

Selection option codes:

SS—Starts the Task Scheduler when selected. Scheduler is invoked as a detached process with the name of CM50_SCHEDULER. It remains active until stopped from this menu.

ST—Stops the Task Scheduler and removes the CM50_SCHEDULER process plus any subprocesses associated with it.

CT—Invokes the system editor and loads the Command Table file for editing.

PT—Purges all but two command table versions. Retains the current table and the immediately previous version.

Choice field:

This field is displayed on the bottom line and accepts the 2-character selection code for the option choice.

Function Keys:

`GPF4/Exit`—Exits the Task Scheduler Menu and returns to the CM50S Main Menu.

`PF2/Help`—See heading 3.1.3.

`PF4/Prior`—Returns to previous menu.

`G-Enter/Main Menu` = Returns to the Main Menu.

The Task Scheduler menu file, CM50_MENU_SCHED.TBL, resides in a directory named CM50$FORMS.

7.1.5 Scheduler DCL Command Files

The Task Scheduler utilizes two DCL command files named CM50_SCHED_START.COM and CM50_SCHED.COM. They reside in the command file directory named CM50$LIB. The command files are invoked by selecting SS (Start Task Scheduler) from the Task Scheduler menu. In turn, the command files invoke Task Scheduler as a detached process with the name of CM50_Scheduler.

7.2 MAKEINC UTILITY

This routine is used to create "include" files that contain arrays of Internal IDs of LCN point.parameters in the form required by these types of data get/store interface routines:

- Point List Transfers—heading 10.2 (FORTRAN), 14.2 (Pascal), or 18.2 (C)
- Raw Data Transfers—heading 10.4 (FORTRAN), 14.4 (Pascal), or 18.4 (C)
- Get/Store Single Point (Internal ID)—headings 10.3.3 and 10.3.4 (FORTRAN), 14.3.3 and 14.3.4 (Pascal), or 18.3.3 and 18.3.4 (C)

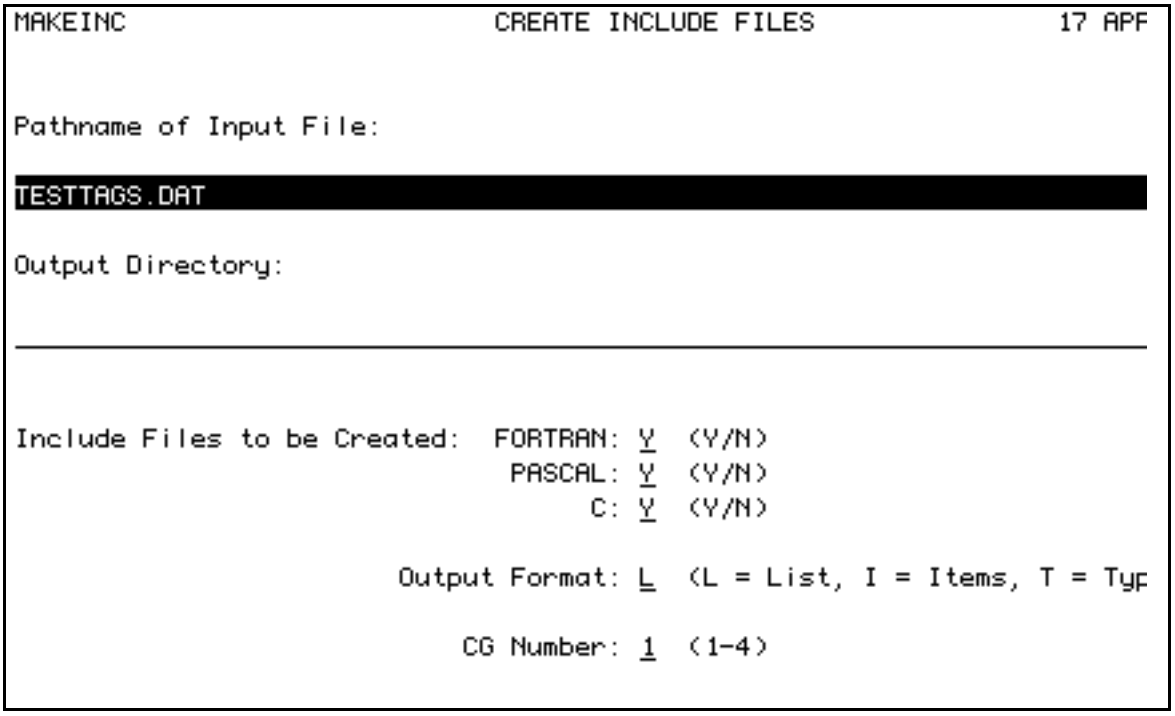
You first prepare an input file that contains a list of all point.parameters to be addressed. This is a text file with one LCN point.parameter name for each line. A sample input file follows.

```
AMCDS.RL0020(1)
AMCDS.RL0020(3)
AMCDS.ASC(2)
AMCDS.ENM(2)
```

Point.parameter name restrictions—The point name can consist of up to sixteen characters, but cannot contain any spaces. If the LCN includes a Network Gateway, the point name may be preceded by a 1- or 2-character LCN identifier (pinid) and a backslash (\) delimiter. The parameter name can be up to eight characters long, but cannot contain any spaces. For an element within an LCN array, the parameter name may be followed by a one to four digit array index enclosed in parentheses. A period must separate the Point and Parameter names. The total length of the Point.parameter entry cannot exceed 40 characters.

7.2.1 Running the Makeinc Program

Once the input file has been prepared, select Makeinc from the CM50S Main Menu. The Create Include Files screen is then displayed.



NOTE

The arrays of internal point.parameter addresses need to be rebuilt and the program(s) using them need to be recompiled whenever the LCN database is changed in a significant manner, such as by the rebuild or deletion of data points referenced in the address array.

The fields on the Create Include Files screen are

- `Full Pathname of Input File`—Enter the VMS pathname of the input file.
- `Output Directory`—Enter the VMS pathname of the output directory. Do not include a file name. If this field is left blank, Makeinc defaults to the current working directory.
- `Include Files to be Created`—Makeinc automatically creates three include files; one each for FORTRAN, for Pascal, and for C. Type N in any of the three fields to stop creation of an include file for that language.

- `Output Format`— Determines whether the include file creates one variable name for all point.parameters of the same value type or creates separate variable names for each point.

  L — List: Creates a named array of internal IDs for each value type. Input is limited to 300 point.parameters in each array. This format is intended for use with the Point List calls.

  I — Items: Creates separate named variables for each point.parameter. This format is intended for use with the Single Point data transfers.

  T — Typed Items: Provides the same output as the Items options, plus adds an additional variable—containing the `val_type` code—for each point.parameter. This option is useful in the preparation of generalized routines where the parameter data type cannot be known and also can be used to verify that the actual data type matches the expected data type (see headings 11.2.1.2, 15.2.1.2, and 19.2.1.2 for `val_type` value assignments).

- `CG Number`—Enter the CG Port number through which the data is to be accessed. The default is port 1.

The following function keys are used to manipulate this screen.

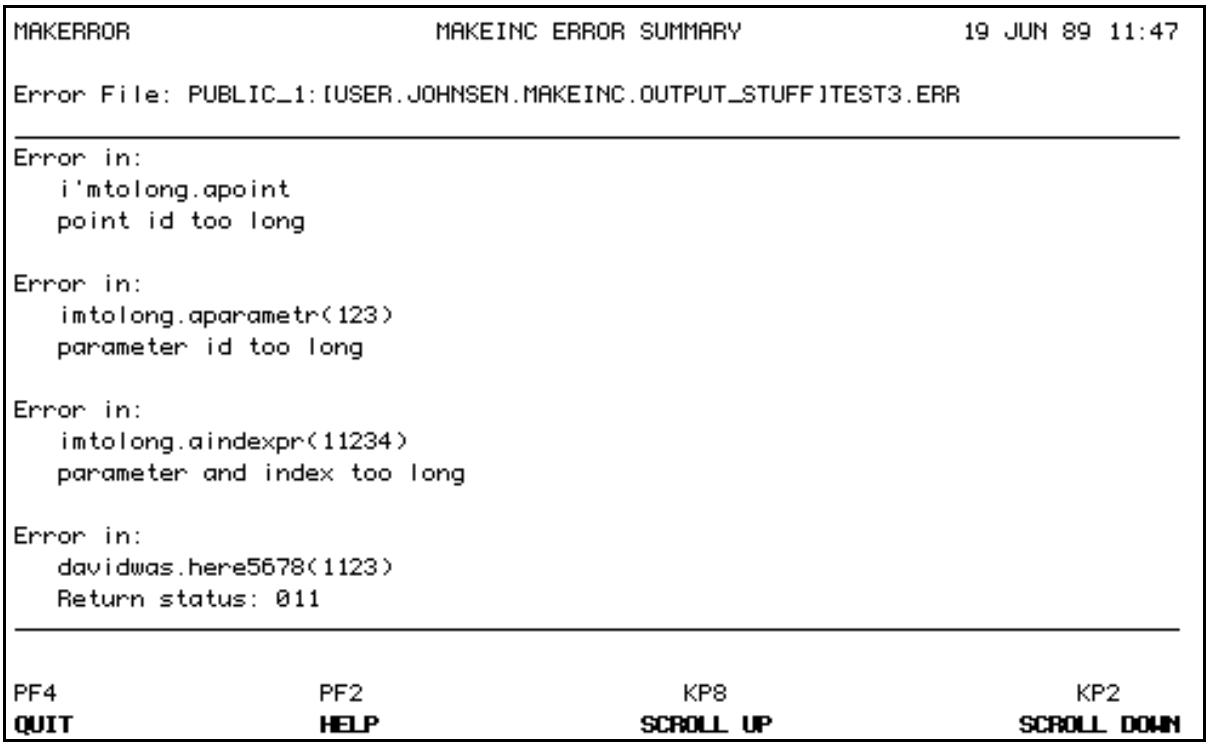
`PF4/QUIT`—Exits the Makeinc utility and returns to the ACP Operations screen.

`PF2/HELP`—See heading 3.1.3.3.

`KP0/ACTIVATE`—Activates the program that creates the specified include files. Each include file has a language-specific extension appended to its name (.INF for FORTRAN, .INP for Pascal, and .INC for C). On completion either a file(s) created message or an error message is displayed on the bottom of the screen.

`KP4/DISPLAY ERRORS`—Displays the Makeinc error file created during processing.

7.2.2 Makeinc Error Screen



Makeinc Error Screen Functions

PF4/QUIT—Exits the Makeinc Error screen and returns to the Create Include Files screen.

PF2/HELP—See heading 3.1.3.

KP8/SCROLL UP—Displays the previous line of the error file. If the first line of the error file is already on display, a message stating so appears at the bottom of the screen.

KP2/SCROLL DOWN—Displays the next line of the error file. If the last line of the error file is already on display, a message stating so appears at the bottom of the screen.

7.2.3 Using the List Format

A shortword integer array is created for each type of parameter listed in the include file. Each array contains a list of internal point id_blocks. You then "include" the .INF, .INP, or .INC file in your source code to make the information available to the compiled program.

If the input file contained any errors, that fact is noted on the first line of the include and the include file will contain references only to the correctly translated point.parameter id_blocks.

The normal first line of the include file is a comment that identifies the number of lists generated and the value types for each. The value type names are truncated to 4 letters: REAL, INTE, ASCI, and ENUM, etc. Additional comment lines follow the declarations for each list to identify the tag names of each point included in the list.

The input file name is used to create variable names for the Internal ID array and size of the list as shown below.

1. The Id_block array variable for each list is generated by appending the input file name (without directory or extension) to a prefix that identifies the value type of all the items in the list. The value type prefixes are:

REAL_	reals
INTE_	integers
ASCI_	ASCII 24-character strings
ENUM_	enumerations
PTID_	internal entity IDs
TIME_	time values
STRI_	40-character strings

2. The shortword variable that is initialized to the number of points in the list is named by adding the prefix CT_ to the Id_block array name.

7.2.3.1 Sample FORTRAN (.INF) Include File

```
*** input contained errors ***
*   3 lists: REAL  ASCI  ENUM
*-------------------------------------------*

       INTEGER*2   CT_REAL_SAMPLE
         DATA      CT_REAL_SAMPLE                 /  2/
       INTEGER*2   REAL_SAMPLE                 (8,  2)
         DATA      REAL_SAMPLE                   /
     -          256,   315,  6912,   258, 21504, 18336,   256,    0,
     -          256,   315,  6912,   258, 21504, 18336,   256,    0 /

* lcn points in this list:
*      1   R_TEST02.NUM(1)
*      2   R_TEST02.NUM(2)
*-------------------------------------------*

       INTEGER*2   CT_ASCI_SAMPLE
         DATA      CT_ASCI_SAMPLE                 /  1/
       INTEGER*2   ASCI_SAMPLE                 (8,  1)
         DATA      ASCI_SAMPLE                   /
     -          768,   315,  6912,   258, 20224, 18592,     0,    0 /

* lcn points in this list:
*      1   R_TEST02.STR
*-------------------------------------------*

       INTEGER*2   CT_ENUM_SAMPLE
         DATA      CT_ENUM_SAMPLE                 /  1/
       INTEGER*2   ENUM_SAMPLE                 (8,  1)
         DATA      ENUM_SAMPLE                   /
     -         1024,   315,  6912,   258, 20224, 18848,     0, 1032 /

* lcn points in this list:
*      1   R_TEST02.CENM
*-------------------------------------------*
```

7.2.3.2 Sample Pascal (.INP) Include File

```
{*** input contained errors ***}
{*   3 lists: real  asci  enum                          *}
{*-----------------------------------------------*}

  var
  CT_REAL_SAMPLE                      : [word] -32768..32767  :=  2;
  REAL_SAMPLE                       : array [1..002]
                         of array [1..8] of [word] -32768..32767 := (
            (   256,   315,  6912,   258, 21504, 18336,   256,    0),
            (   256,   315,  6912,   258, 21504, 18336,   256,    0) );

{* lcn points in this list:
{*     1   R_TEST02.NUM(1)
{*     2   R_TEST02.NUM(2)
{*-----------------------------------------------*}

  var
  CT_ASCI_SAMPLE                      : [word] -32768..32767  :=  1;
  ASCI_SAMPLE                       : array [1..001]
                         of array [1..8] of [word] -32768..32767 := (
            (   768,   315,  6912,   258, 20224, 18592,     0,    0) );

{* lcn points in this list:
{*     1   R_TEST02.STR
{*-----------------------------------------------*}

  var
  CT_ENUM_SAMPLE                      : [word] -32768..32767  :=  1;
  ENUM_SAMPLE                       : array [1..001]
                         of array [1..8] of [word] -32768..32767 := (
            (  1024,   315,  6912,   258, 20224, 18848,     0, 1032) );

{* lcn points in this list:
{*     1   R_TEST02.CENM
{*-----------------------------------------------*}
```

7.2.3.3 Sample "C" (.INC) Include File

```
/*** input contained errors ***/
/*   3 lists: real  asci  enum                          */
/*-----------------------------------------------*/

       short int ct_real_SAMPLE                              =  2;
       short int real_SAMPLE                        [  2][8] = {
                256,   315,  6912,   258, 21504, 18336,   256,    0,
                256,   315,  6912,   258, 21504, 18336,   256,    0 };

/* lcn points in this list:
/*     1   R_TEST02.NUM(1)
/*     2   R_TEST02.NUM(2)
/*-----------------------------------------------*/

       short int ct_asci_SAMPLE                              =  1;
       short int asci_SAMPLE                        [  1][8] = {
                768,   315,  6912,   258, 20224, 18592,     0,    0 };

/* lcn points in this list:
/*     1   R_TEST02.STR
/*-----------------------------------------------*/

       short int ct_enum_SAMPLE                              =  1;
       short int enum_SAMPLE                        [  1][8] = {
               1024,   315,  6912,   258, 20224, 18848,     0, 1032 };

/* lcn points in this list:
/*     1   R_TEST02.CENM
/*-----------------------------------------------*/
```

7.2.4 Using the Item and Typed Item Formats

For the Item and Typed Item formats, a shortword integer array is created for each point.parameter listed in the include file. Each array contains the point.parameter's `id_block` (and, optionally, the parameter's `val_typ)`. You then "include" the `.INF`, `.INP`, or `.INC` file in your source code to make the information available to the compiled program.

The point and parameter names specified in the input file are used to create variable names for the Internal ID and Value Type as shown below.

1. If the point name begins with a number, the three characters “CM_” are placed in front of the number.

2. The period used to separate the point name from the parameter name is converted to a double underscore ('__'). For example, `AMCDS.ASC` is converted to `AMCDS__ASC.`

3. Parentheses indicating an index of an array are changed to a pair of dollar signs ($) surrounding the index value. For example, `AMCDS.RL(1)` is converted to `AMCDS__RL$1$`. Note that a negative array subscript generates a single underbar character preceding the index value. For example, `AMCDS.RL(-1)` is converted to `AMCDS__RL$_1$`.

4. If type information has been included, the variable containing the type is the same as the ID block variable, but has `_TYPE` appended to it. For example, `AMCDS__RL$1$_TYPE.`

5. If the tagname includes a Network pinid, the backslash delimiter is converted to a dollar sign ($).

7.2.4.1 Sample FORTRAN (.INF) Include File

```
   INTEGER*2           AMCDS__RL0020$1$(8)
   DATA                AMCDS__RL0020$1$
-                      /     256,    278,   3840,    514,
-                          -26880,   8352,    256,      0/

   INTEGER*2           AMCDS__RL0020$1$_TYPE
   DATA                AMCDS__RL0020$1$_TYPE          /       1/

   INTEGER*2           AMCDS__RL0020$3$(8)
   DATA                AMCDS__RL0020$3$
-                      /     256,    278,   3840,    514,
-                          -26880,   8352,    768,      0/

   INTEGER*2           AMCDS__RL0020$3$_TYPE
   DATA                AMCDS__RL0020$3$_TYPE          /       1/

   INTEGER*2           AMCDS__ASC$2$(8)
   DATA                AMCDS__ASC$2$
-                      /     768,    278,   3840,    514,
-                          -26880,   2976,    512,      0/

   INTEGER*2           AMCDS__ASC$2$_TYPE
   DATA                AMCDS__ASC$2$_TYPE             /       3/

   INTEGER*2           AMCDS__ENM$2$(8)
   DATA                AMCDS__ENM$2$
-                      /    1024,    278,   3840,    514,
-                          -26880,   2464,    512,   -505/

   INTEGER*2           AMCDS__ENM$2$_TYPE
   DATA                AMCDS__ENM$2$_TYPE             /       4/
```

7.2.4.2 Sample Pascal (.INP) Include File

```
VAR
  AMCDS__RL0020$1$          : ARRAY[1..8] OF [WORD] -32768..32767;
  AMCDS__RL0020$1$_TYPE     : [WORD] -32768..32767;

VALUE
  AMCDS__RL0020$1$[1]        :=     256;
  AMCDS__RL0020$1$[2]        :=     278;
  AMCDS__RL0020$1$[3]        :=    3840;
  AMCDS__RL0020$1$[4]        :=     514;
  AMCDS__RL0020$1$[5]        :=  -26880;
  AMCDS__RL0020$1$[6]        :=    8352;
  AMCDS__RL0020$1$[7]        :=     256;
  AMCDS__RL0020$1$[8]        :=       0;
  AMCDS__RL0020$1$_TYPE      :=       1;

VAR
  AMCDS__RL0020$3$          : ARRAY[1..8] OF [WORD] -32768..32767;
  AMCDS__RL0020$3$_TYPE     : [WORD] -32768..32767;

VALUE
  AMCDS__RL0020$3$[1]        :=     256;
  AMCDS__RL0020$3$[2]        :=     278;
  AMCDS__RL0020$3$[3]        :=    3840;
  AMCDS__RL0020$3$[4]        :=     514;
  AMCDS__RL0020$3$[5]        :=  -26880;
  AMCDS__RL0020$3$[6]        :=    8352;
  AMCDS__RL0020$3$[7]        :=     256;
  AMCDS__RL0020$3$[8]        :=       0;
  AMCDS__RL0020$3$_TYPE      :=       1;

VAR
  AMCDS__ASC$2$             : ARRAY[1..8] OF [WORD] -32768..32767;
  AMCDS__ASC$2$_TYPE        : [WORD] -32768..32767;

VALUE
  AMCDS__ASC$2$[1]           :=     768;
  AMCDS__ASC$2$[2]           :=     278;
  AMCDS__ASC$2$[3]           :=    3840;
  AMCDS__ASC$2$[4]           :=     514;
  AMCDS__ASC$2$[5]           :=  -26880;
  AMCDS__ASC$2$[6]           :=    2976;
  AMCDS__ASC$2$[7]           :=     512;
  AMCDS__ASC$2$[8]           :=       0;
  AMCDS__ASC$2$_TYPE         :=       3;

VAR
  AMCDS__ENM$2$             : ARRAY[1..8] OF [WORD] -32768..32767;
  AMCDS__ENM$2$_TYPE        : [WORD] -32768..32767;

VALUE
  AM001__ENM$2$[1]          :=     512;
  AM001__ENM$2$[2]          :=     276;
  AM001__ENM$2$[3]          :=    3840;
  AM001__ENM$2$[4]          :=       2;
  AM001__ENM$2$[5]          :=   19456;
  AM001__ENM$2$[6]          :=   31753;
  AM001__ENM$2$[7]          :=       0;
  AM001__ENM$2$[8]          :=       0;
  AM001__ENM$2$_TYPE        :=       2;
```

7.2.4.3 Sample "C" (.INC) Include File

```
SHORT INT AMCDS__RL0020$1$[8]        = {     256,    278,   3840,    514,
-                                   -26880,   8352,    256,      0};
SHORT INT AMCDS__RL0020$1$_TYPE      = {       1};

SHORT INT AMCDS__RL0020$3$[8]        = {     256,    278,   3840,    514,
-                                   -26880,   8352,    768,      0};
SHORT INT AMCDS__RL0020$3$_TYPE      = {       1};

SHORT INT AMCDS__ASC$2$[8]           = {     768,    278,   3840,    514,
-                                   -26880,   2976,    512,      0};
SHORT INT AMCDS__ASC$2$_TYPE         = {       3};

SHORT INT AMCDS__ENM$2$[8]           = {    1024,    278,   3840,    514,
-                                   -26880,   2464,    512,   -505};
SHORT INT AMCDS__ENM$2$_TYPE         = {       4};
```

7.3 LCN FILE TRANSFER OPERATIONS

This facility allows execution of LCN File Transfer functions from a VAX terminal, including the archiving of LCN Files on the VAX.

The file transfer operations are supported only by LCNs running TDC 3000 release 400 or later. Also, concurrent execution of file transfer requests is limited to one user on each CG port. When a File Transfer session is initiated the CG port number is defaulted to 1, but once the user has specified a CG port number the specified CG is used for any subsequent operations within the session (until it is explicitly changed by the user).

The DATAOUT facility allows the user, when requesting the execution of specific file transfer transactions, to place relevant data in the dataout or catalog file. This dataout file is a shared file for all users of file transfer on the LCN. For example, user "Jones" executes a List Volume Names request, the results of which are placed into the current dataout file. User "Smith" then requests a List File Attributes. These results also are placed into the same (current) dataout file.

List Volume Names and List File Attributes are the only file transfer operations that require a dataout file. Other file transfer transactions treat dataout as an option that can be used to log file copies, moves, renames and deletes. For any operation that allows the use of dataout, the file currently assigned for dataout in the specified CG is displayed (and automatically changed when the CG selection is changed). Generally, the dataout file should be given an extension (like ".X") that identifies it as a User Text file on the LCN.

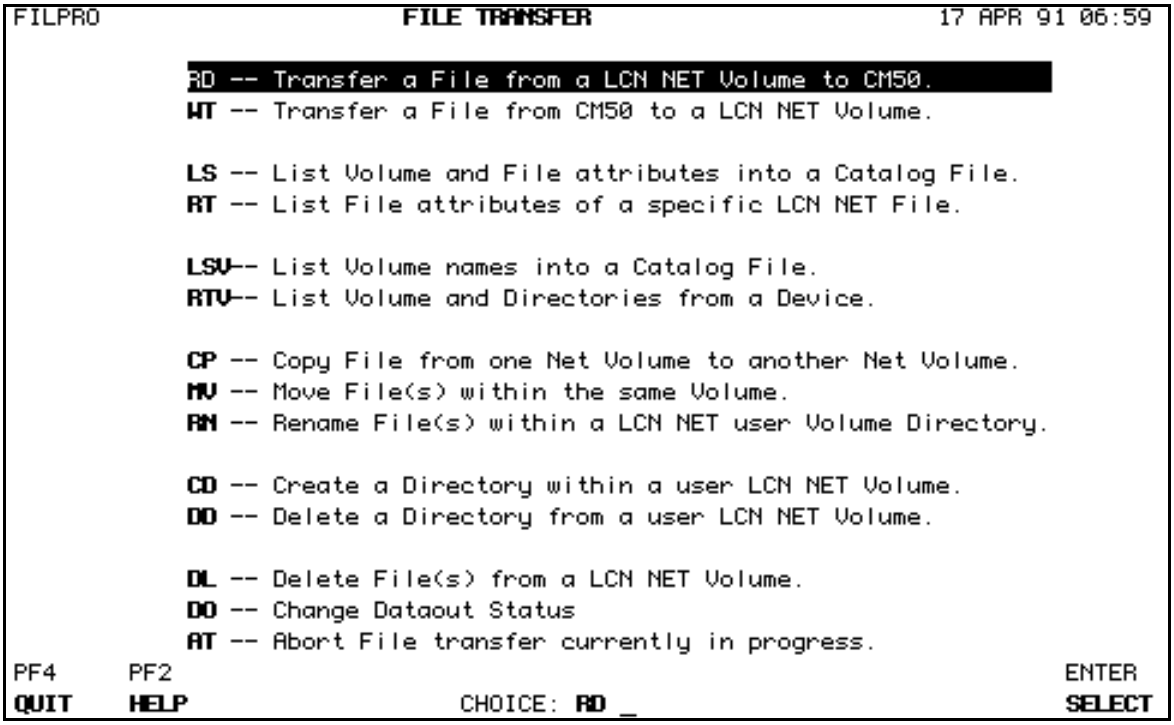
An LCN ATTRIBUTES file is created on the VAX (with an extension of .LA) whenever an LCN file is read into the VAX. This file is always newly created (since a unique file name is required for the file READ) and therefore should always be version 1. The purpose of the LCN attributes file is two-fold.

- The unique LCN attributes such as file descriptor are kept here as are the actual attributes of record size, block size, etc.

- The creation date, version number and other parameters are kept to insure that binary files are not modified and then written back to the LCN NET volume. The Attributes file is used to format the file sent to the LCN by the file WRITE requests. Part of this verification prevents modified binary files from being written to the LCN.

7.3.1 File Transfer Menu

All File Transfer processes can be invoked through the File Transfer Menu selection on the CM50S Main Menu. Select the desired function by typing the selection code or by using the arrow keys to tab through the menu until the desired choice is highlighted, then press <RETURN>.



7.3.1.1 File Transfer Choices

Each of the file transfer choices brings up a data entry form with a unique function.

- RD—Transfers a single file from the LCN NET volume to CM50S.
- WT—Transfers a single file from CM50S to the LCN NET volume.
- LS—Lists into a specified catalog or dataout file the attributes of one or more LCN NET resident files. The catalog file of results can be transferred to CM50S using the RD process.
- RT—Displays at the terminal the LCN file attributes of a specific LCN NET file.
- LSV—Lists into a specified catalog or dataout file the LCN NET VOLUME names and directories of one or all History Modules.
- RTV—Displays at the terminal the volume names, directory names, and sector usage of a specific History Module.
- CP—Copies one or more LCN NET files from one NET volume to another NET volume.

• MV—Moves one or more LCN NET files from one directory to another, within the same LCN NET volume.

• RN—Permits the renaming of one or more LCN NET files within a LCN NET volume directory.

• CD—Creates a directory under a volume on the History Module.

• DD—Deletes a directory from under a volume on the History Module.

• DL—Deletes one or more LCN NET files from the specified volume on the History Module. Once deleted the files cannot be recovered.

• DO—Manipulates the dataout or catalog file. The file can be created and opened or closed. The file can be deleted using the DL process.

• AT—Aborts the file transfer operation that is currently in process.

7.3.1.2 File Transfer Menu Function Keys

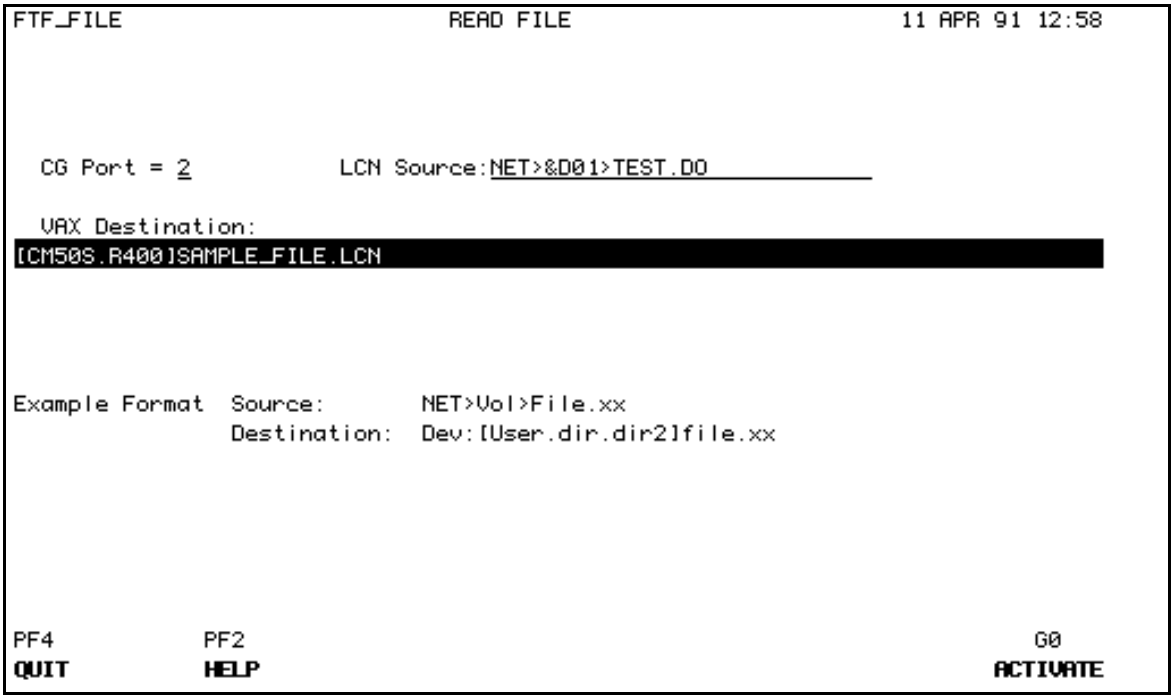
PF4/QUIT—Exit File Transfer, return to CM50S Main Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

ENTER/SELECT—Displays the form for the selected file transfer function The selected function is identified in the CHOICE: field and is highlighted.

7.3.2 Read File From LCN

The RD menu option provides a screen to read a file from the LCN and store it in the VAX.



7.3.2.1 Read File Fields

The READ FILE fields are:

• CG Port—Number of the CG Port (1- 4) to be used for the file transfer request.

• LCN SOURCE—LCN file pathname of the file to be read. The device may be specified either by node number (PN:nn>) or as NET>. If the LCN includes a Network Gateway, remote nodes can be addressed by including a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

Example pathname formats: NET>&D01>FILENAME.DO
ab\NET>&D01>FILENAME.DO

• VAX DESTINATION—VMS pathname of the recipient file, including an extension (but not version number). The volume and directory names default to the user's current directory. If no extension is specified, the VMS default of .DAT is used. This pathname also controls where the companion LCN attributes file will be stored. The LCN attributes file uses the following naming convention: The original extension becomes part of the filename preceded by an underscore (_), and a new extension of ".LA" is appended. For example, the VAX destination of FORMULAE.CL would have a companion attributes file of FORMULAE_CL.LA.

NOTE

In order to maintain consistency of attributes and data files, the VAX Destination cannot match the VMS pathname of an existing file.

7.3.2.2 Read File Function Keys

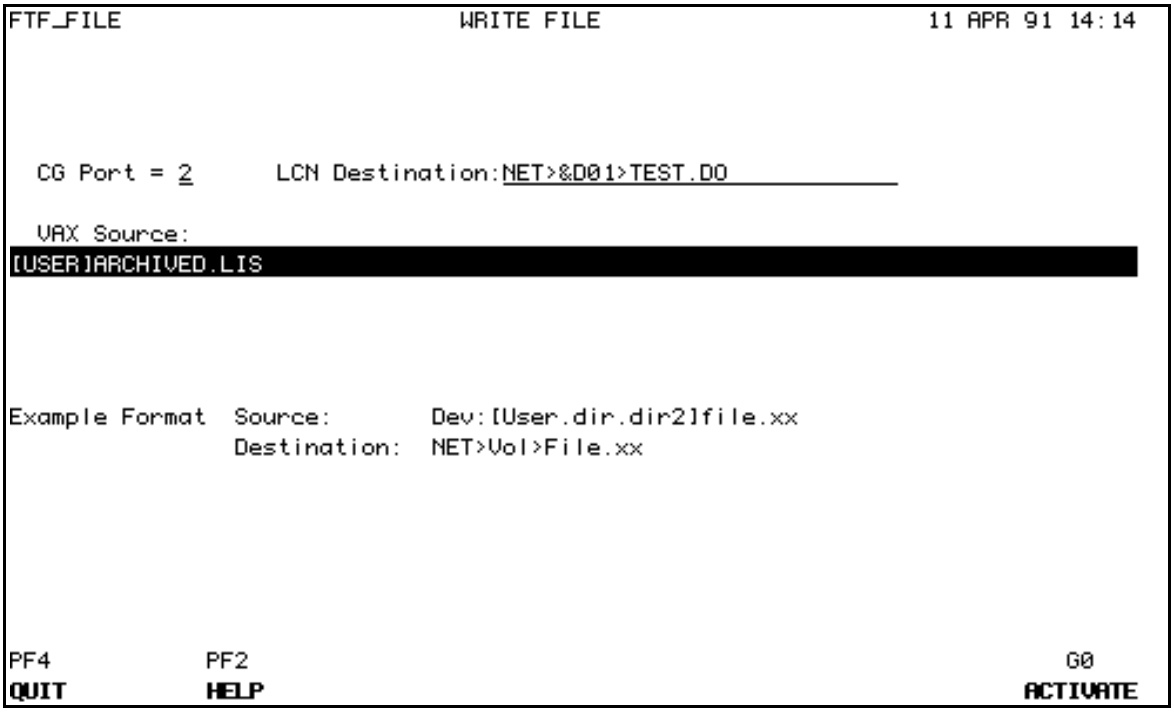
PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Starts the file transfer process. Error and Information messages will be displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager also return a secondary error code that identifies the specific error (see Appendix A.4 for translation). The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>, will clear the error condition and allow for further processing.

7.3.3 Write File to LCN

The WT menu option provides a screen used to copy a file that has been archived on the VAX back to the LCN. If the LCN file exists, it is replaced without warning or error indication. (This is unlike the programmatic interface to File Transfer, which requires the user to select whether the write transaction should be aborted or allowed if the file to be written already exists at the LCN destination.) Any file that has been read from an LCN can be written back to that same LCN, but only ASCII files can be modified on the VAX and then written to an LCN.



7.3.3.1 Write File Fields

The WRITE FILE fields are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• VAX SOURCE—VMS pathname of the file to be copied to the LCN. This file must also have a companion LCN attributes file. The LCN attributes file uses the following naming convention: The original extension becomes part of the filename preceded by an underscore (_), and a new extension of ".LA" is appended. For example, the VAX destination of FORMULAE.CL would have a companion attributes file named FORMULAE_CL.LA.

• LCN DESTINATION—LCN file pathname of the file to be written. The device may be specified either by node number (PN:nn>) or as NET>. If the LCN includes a Network Gateway, remote nodes can be addressed by including a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

Example pathname formats: NET>&D01>FILENAME.DO
ab\NET>&D01>FILENAME.DO

7.3.3.2 Write File Function Keys

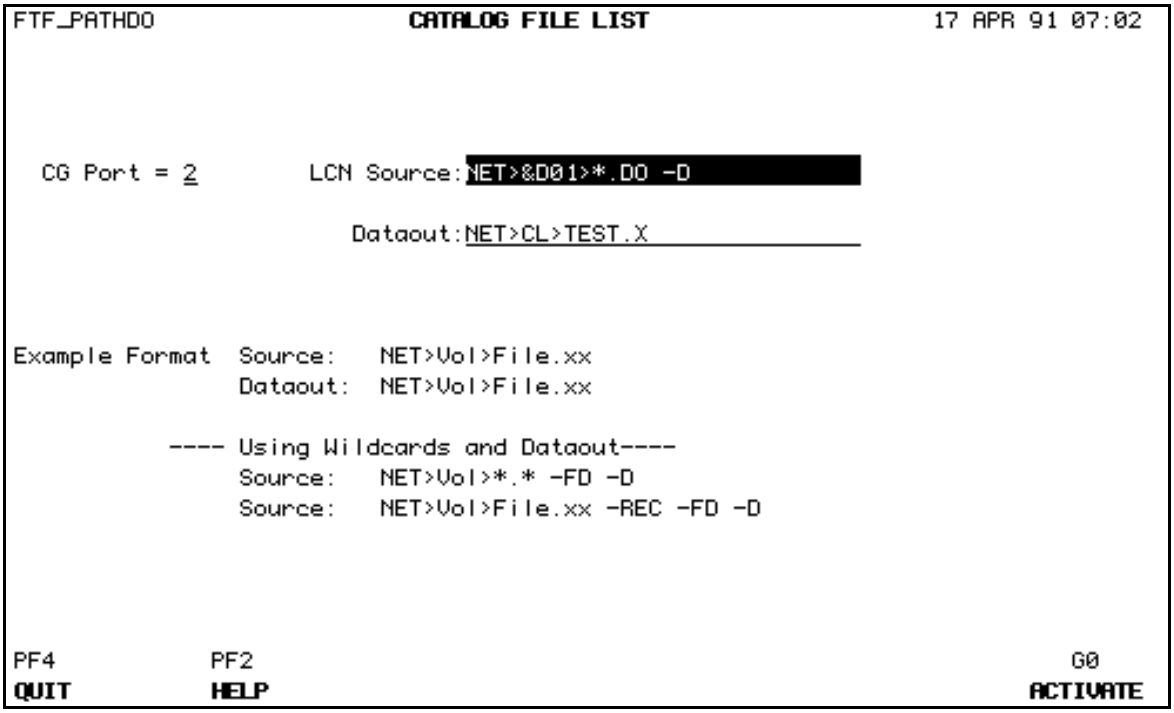
PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Starts the file transfer process. Error and Information messages will be displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager also return a secondary error code that identifies the specific error (see Appendix A.4 for translation). The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>, will clear the error condition and allow for further processing.

7.3.4 Catalog File List

Use the LS menu option to capture the file attributes of one or more files into a dataout or catalog file.



7.3.4.1 Catalog File List Fields

The FILE LIST fields are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• LCN Source—LCN file pathname of the files to be listed. The device may be specified either by node number (PN:nn>) or as NET>. Wildcards (*) may be used for the filename and/or extension. Example format: NET>&D01>*.DO. Additional data may be included in the catalog file by specifying one or more of the following options at the end of the pathname:

-FD include file descriptors
-REC include record/block information

If the LCN includes a Network Gateway, remote nodes can be addressed by adding a prefix—consisting of the one- or two-character "pinid" followed by a backslash (\)—to the pathname.

• Dataout—LCN pathname of dataout file to be used as a journal/cataloging file using form: NET>&DIR>FILENAME.xx

7.3.4.2 Catalog File List Function Keys

PF4/QUIT—Return to the File Transfer Menu.

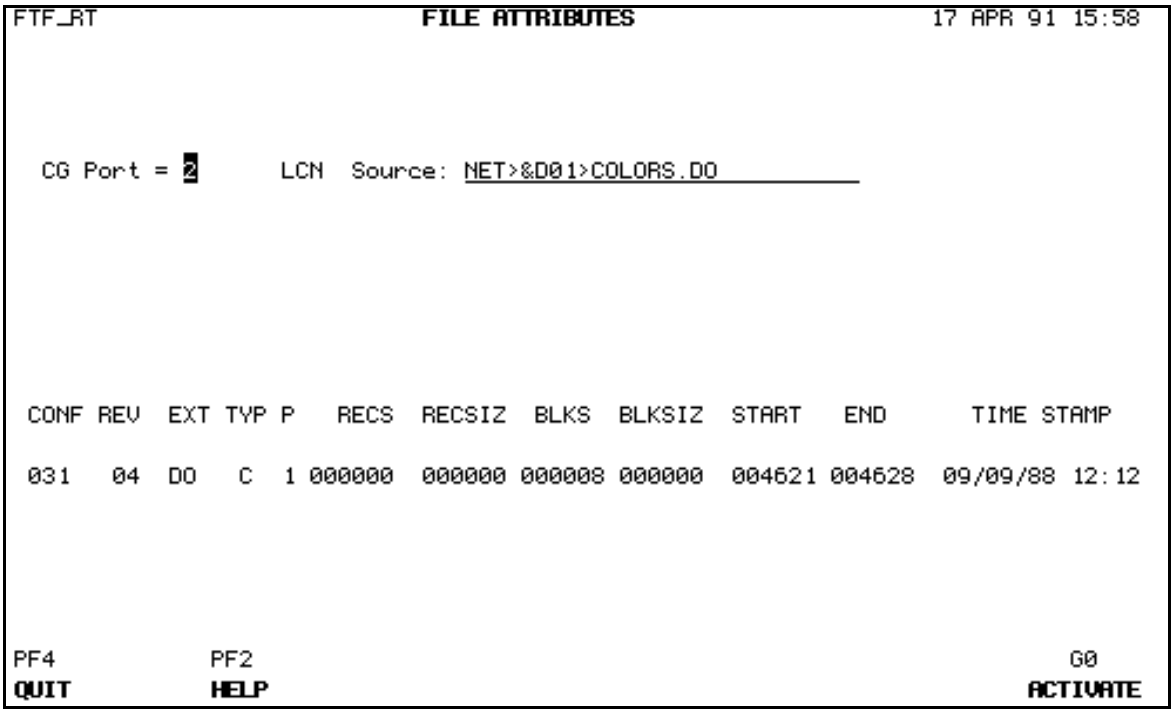
PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Starts the file list process. Error and Information messages will be displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager also return a secondary error code that identifies the specific error (see Appendix A.4 for translation). The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>, will clear the error condition and allow for further processing.

7.3.5 File Attributes

Use the RT menu option to retrieve the file attributes for a specific LCN file. Wildcard characters are not permitted. The following attributes will be displayed:

LABEL	ATTRIBUTE
• CONF	LCN File Configuration
• REV	LCN File Revision
• EXT	File Extension
• TYP	LCN File Type: C=contiguous, L=linked
• P	LCN File Protection: 0=no, 1=yes
• RECS	Logical number of records
• RECSIZ	Record size
• BLKS	Logical number of blocks
• BLKSIZ	Block size
• START	Starting Sector
• END	Ending Sector
• TIME STAMP	Timestamp (MM/DD/YY hh:mm)
•	File Descriptor (optional)



7.3.5.1 File Attributes Fields

The FILE ATTRIBUTES fields are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• LCN Source—LCN pathname of the file whose attributes are to be displayed. The device may be specified either by node number (PN:nn>) or as NET>. If the LCN includes a Network Gateway, remote nodes can be addressed by adding a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

Example pathname formats: NET>&D01>FILENAME.DO
ab\NET>&D01>FILENAME.DO

If the option " -FD" is appended to the path, then the File Descriptor will be displayed.

7.3.5.2 File Attributes Function Keys

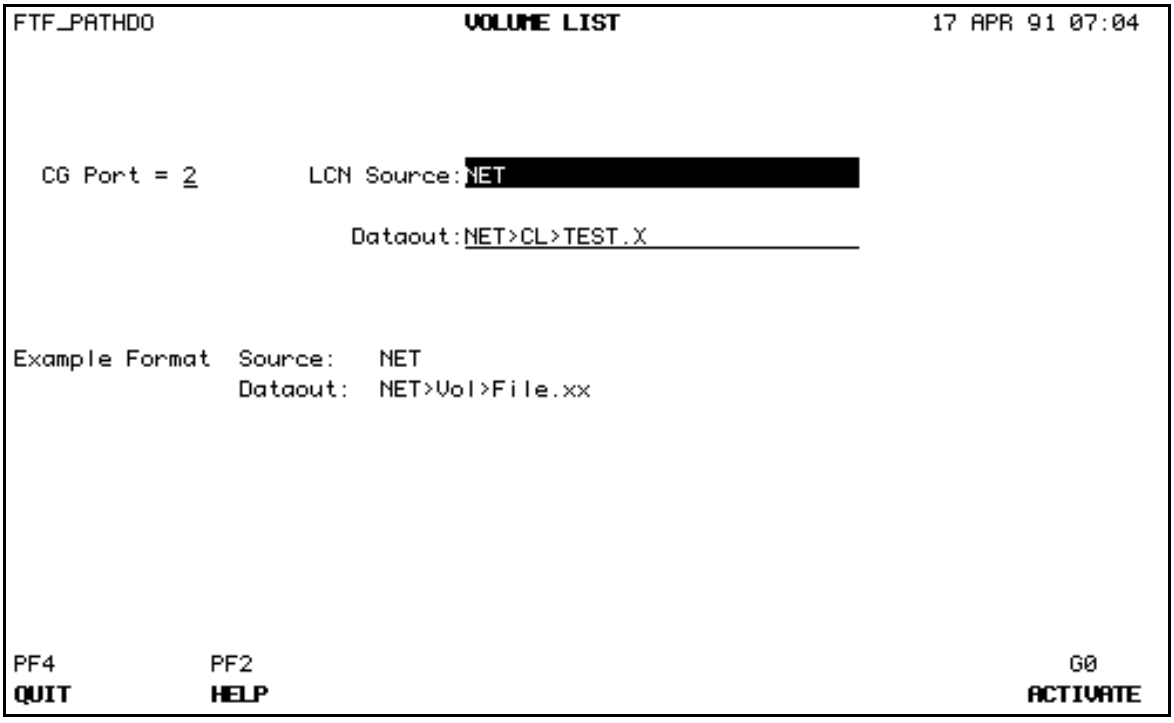
PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Requests the file attributes. If the request is successful, the attributes will be displayed on the same screen, otherwise Error and Information messages will be displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager also return a secondary error code that identifies the specific error (see Appendix A.4 for translation). The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>, will clear the error condition and allow for further processing.

7.3.6 Volume List

The LSV menu option provides a screen to capture the Volume and Directories of one or more History Modules into a dataout or catalog file.



7.3.6.1 Volume List Fields

The VOLUME LIST fields are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• LCN Source—LCN file pathname using the **required** form: NET.

• Dataout—LCN pathname of dataout file to be used as a journal/cataloging file using form: NET>&DIR>FILENAME.xx

7.3.6.2 Volume List Function Keys

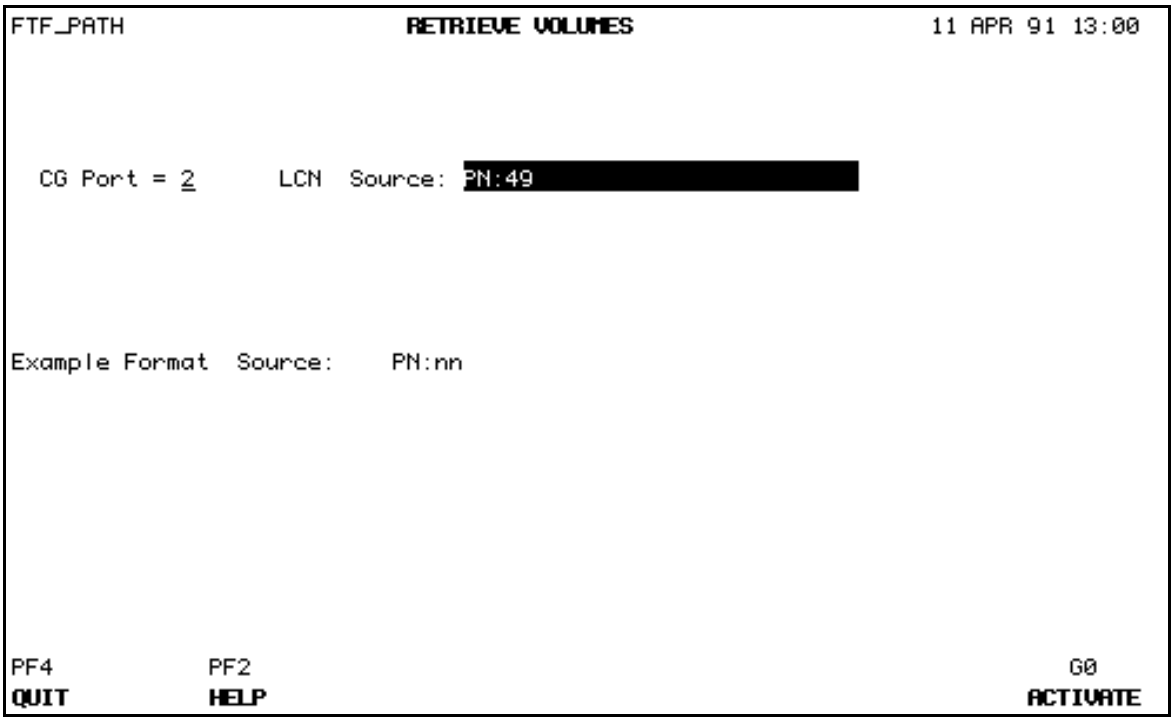
PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Starts the file transfer process. Error and Information messages will be displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager also return a secondary error code that identifies the specific error (see Appendix A.4 for translation). The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>, will clear the error condition and allow for further processing.

7.3.7 Retrieve Volumes

The RTV menu option provides a screen to display all the Volume and Directory names on a History Module. Successful completion of the request will bring up a secondary display that allows the user to scroll through the volumes.



7.3.7.1 Retrieve Volumes Fields

The RETRIEVE VOLUMES fields are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• LCN Source—LCN node name using the **required** form PN:nn, where nn is the node number of the History Module to be queried**.**

7.3.7.2 Retrieve Volumes Function Keys

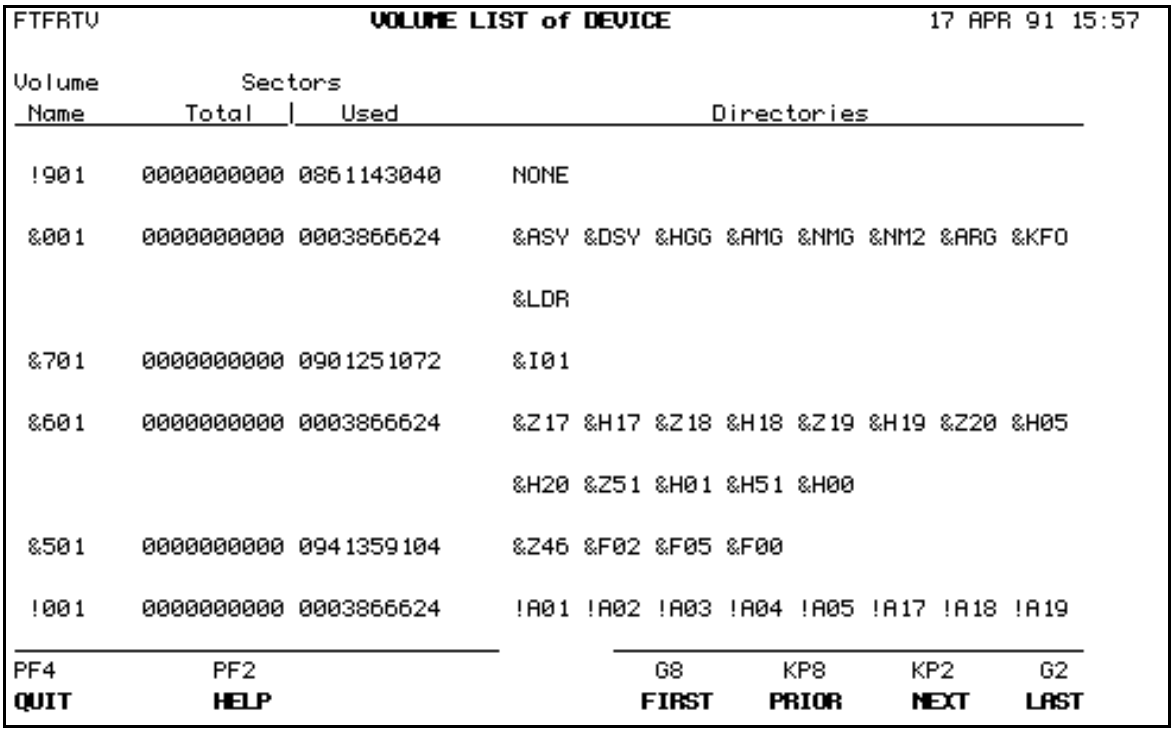
PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Starts the Retrieve Volumes process. If the information is successfully retrieved, it will appear on the following display; otherwise, Error and Information messages are displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager also return a secondary error code that identifies the specific error (see Appendix A.4 for translation). The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>, will clear the error condition and allow for further processing.

7.3.7.3 Volumes & Directories Display

This screen displays the returned volumes and the directories that exist within them.



The following function keys are active:

PF4/QUIT—Return to the Retrieve Volumes Selection screen.

PF2/HELP—Provides help information -- see Section 3.1.3.

KP2/NEXT—Scroll forward one volume.

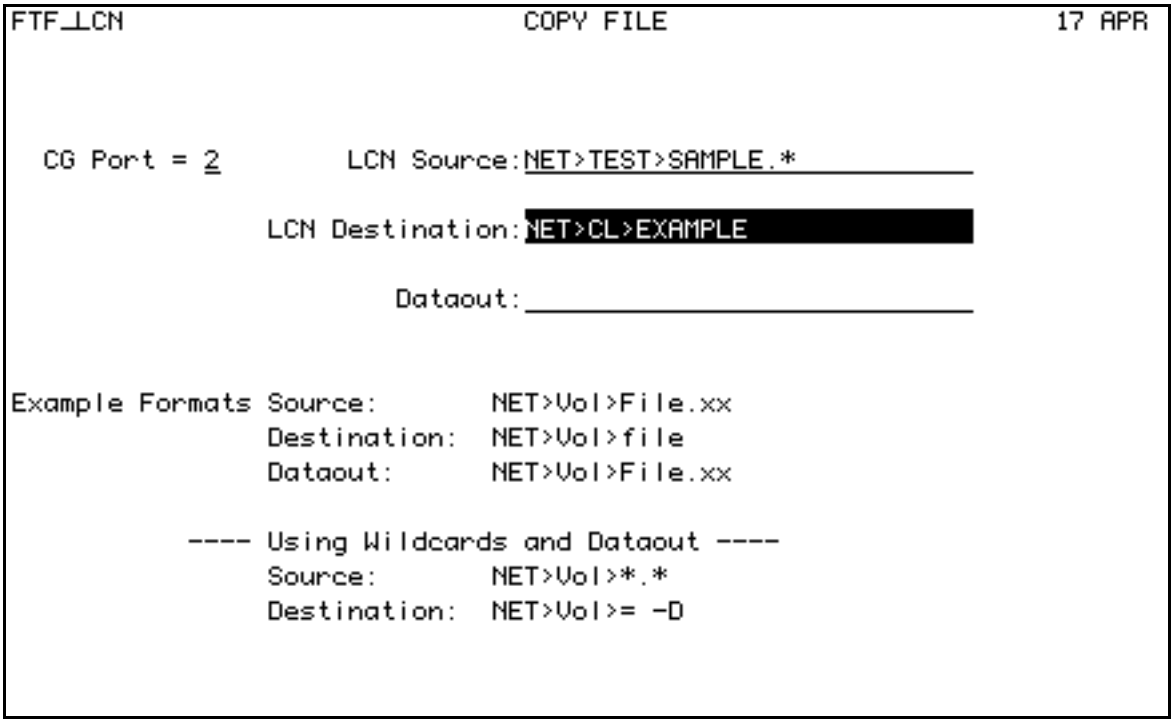
KP8/PRIOR—Scroll backward one volume.

G2/LAST —Jump to display the last volume in the list.

G8/FIRST—Jump to display the start of the list.

7.3.8 Copy File

The CP menu option provides a screen to copy a file from one LCN NET directory to another or to a different filename (but the extension must remain the same). Wildcards permit copying multiple files. Optionally, the actions can be journalized to a dataout file.



7.3.8.1 Copy File Fields

The COPY FILE parameters are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• LCN Source—LCN pathname of the file to be copied. The device may be specified either by node number (PN:nn>) or as NET>. If the LCN includes a Network Gateway, remote nodes can be addressed by adding a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

Example pathname formats: NET>&D01>FILENAME.DO
ab\NET>&D01>FILENAME.DO

Wildcards (*) are permitted for the filename and/or extension.

• LCN Destination—LCN pathname of the new copy of the file. If the LCN includes a Network Gateway, remote nodes can be addressed by adding a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

Including a suffix of " -D" will cause the actions to be journalized to Dataout.

• Dataout—LCN pathname of dataout file to be used as a journal/cataloging file using form: NET>&dir>filename.DO if the " -D" option is specified on the end of the LCN Source pathname. **Failure to specify the -D option will not produce an error message and the file copies will not be journaled.** Using the -D option with a blank (unassigned) dataout will result in an error. The dataout assignment can be changed by typing in the desired pathname on this screen. If a dataout file is changed, the dataout assignment request will be executed before the files are copied.

7.3.8.2 Copy File Function Keys

PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Starts the COPY FILE process using the CG Port and LCN Source parameters provided. If the dataout path is defined and the -D option is specified in the destination pathname, then journal the copied files to the dataout file. Error and Information messages will be displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager also return a secondary error code that identifies the specific error (see Appendix A.4 for translation). The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>, will clear the error condition and allow for further processing.

7.3.9 Move File

Use the MV menu option to move a file from one directory to another within the same Net volume. Neither the volume nor the filename can be changed during the move. Wildcards permit renaming multiple files. Optionally, the actions can be journalized to a dataout file.

```
FTF_LCN                          MOVE FILE                    17 APR 91 07:11


  CG Port = 2          LCN Source:NET>TEST>SAMPLE.*
                  LCN Destination:CL
                          Dataout:


Example Formats Source:       NET>Dir1>File.xx
                Destination:  Dir2
                Dataout:      NET>Vol>File.xx

           ---- Using Wildcards and Dataout ----
                Source:       NET>Dir1>*.*
                Destination:  Dir2   -D



PF4           PF2                                                    G0
QUIT          HELP                                                   ACTIVATE
```

7.3.9.1 Move File Fields

The MOVE FILE parameters are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• LCN Source—LCN pathname of the file to be moved. The device may be specified either by node number (PN:nn>) or as NET>. Example format: NET>&D01>FILENAME.DO. Wildcards (*) are permitted for the filename and/or extension.

• LCN Destination— Name of the LCN directory where the file(s) will reside after the move. Do not include either a device prefix or filename. Including a suffix of "-D" will cause the actions to be journalized to Dataout.

• Dataout—LCN pathname of dataout file to be used as a journal/cataloging file using form: NET>&dir>filename.DO if the " -D" option is specified on the end of the LCN Source pathname. **Failure to specify the -D option will not produce an error message and the file moves will not be journaled.** Using the -D option with a blank (unassigned) dataout will result in an error. The dataout assignment can be changed by typing in the desired pathname on this screen. If a dataout file is changed, the dataout assignment request will be executed before the files are moved.

7.3.9.2 Move File Function Keys

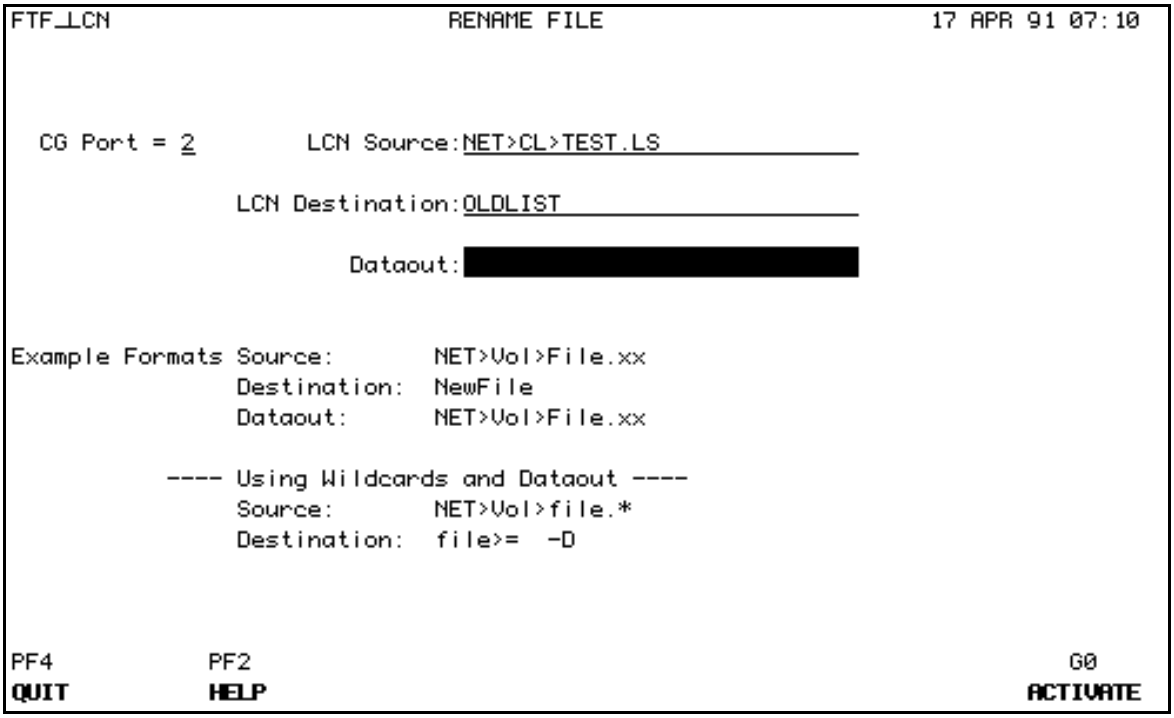
PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Starts the MOVE FILE process using the CG Port and LCN Source parameters provided. If the dataout path is defined and the -D option is specified in the destination pathname, then journal the file moves to the dataout file. Error and Information messages will be displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager also return a secondary error code that identifies the specific error (see Appendix A.4 for translation). The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>, will clear the error condition and allow for further processing.

7.3.10 Rename File

The RN menu option provides a screen to rename a file on the History Module. Wildcards permit renaming multiple files. Optionally, the actions can be journalized to a dataout file.



7.3.10.1 Rename File Fields

The RENAME FILE parameters are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• LCN Source—LCN pathname of the file to be moved. The device may be specified either by node number (PN:nn>) or as NET>. Example format: NET>&D01>FILENAME.DO. Wildcards (*) are permitted for the filename and/or extension.

• LCN Destination— New name of the LCN file. Do not include either a directory or extension specification. Including a suffix of " -D" will cause the actions to be journalized to Dataout.

• Dataout—LCN pathname of dataout file to be used as a journal/cataloging file using form: NET>&dir>filename.DO if the " -D" option is specified on the end of the LCN Source pathname. **Failure to specify the -D option will not produce an error message and the file renames will not be journaled.** Using the -D option with a blank (unassigned) dataout will result in an error. The dataout assignment can be changed by typing in the desired pathname on this screen. If a dataout file is changed, the dataout assignment request will be executed before the files are renamed.

7.3.10.2 Rename File Function Keys

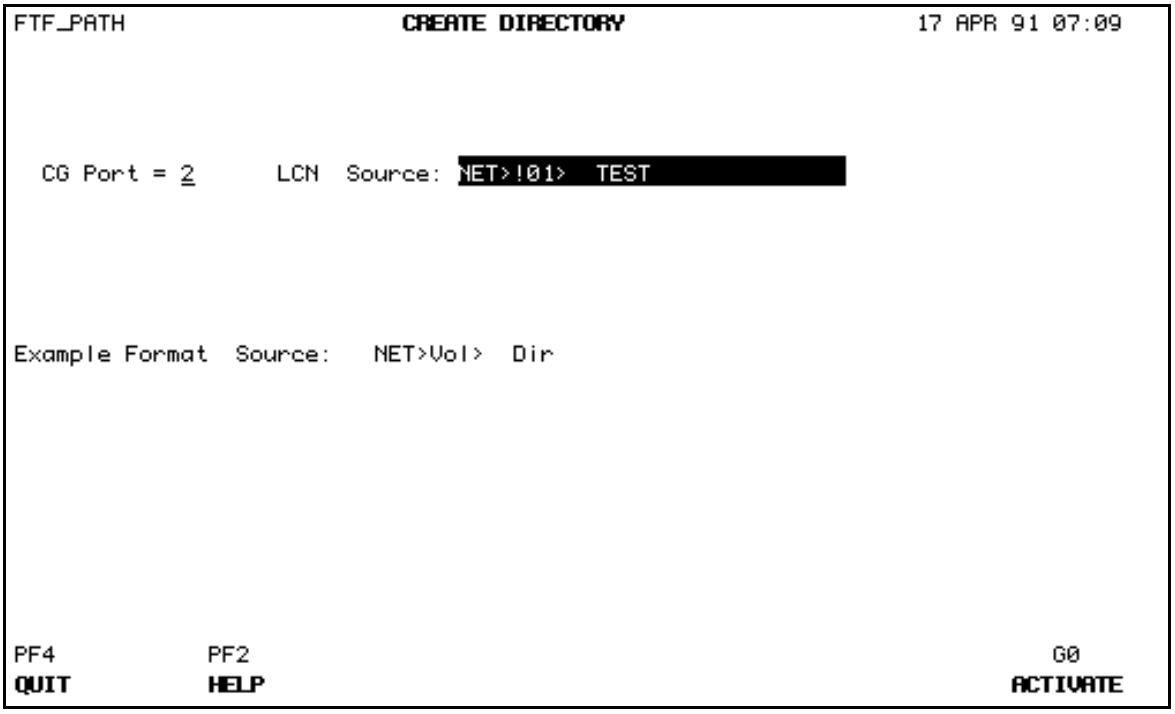
PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Starts the RENAME FILE process using the CG Port and LCN Source parameters provided. If the dataout path is defined and the -D option is specified in the destination pathname, then journal the file renames to the dataout file. Error and Information messages will be displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager also return a secondary error code that identifies the specific error (see Appendix A.4 for translation). The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>, will clear the error condition and allow for further processing.

7.3.11 Create Directory

The CD menu option provides a screen to create a directory under a volume on the History Module. No wildcard characters or dataout options are applicable.



7.3.11.1 Create Directory Fields

The CREATE DIRECTORY fields are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• LCN Source—LCN pathname of the directory to be created. The device may be specified either by node number (PN:nn>) or as NET>. Example format: NET>VOL> DIR. Note the space delimiter before the Directory name.

7.3.11.2 Create Directory Function Keys

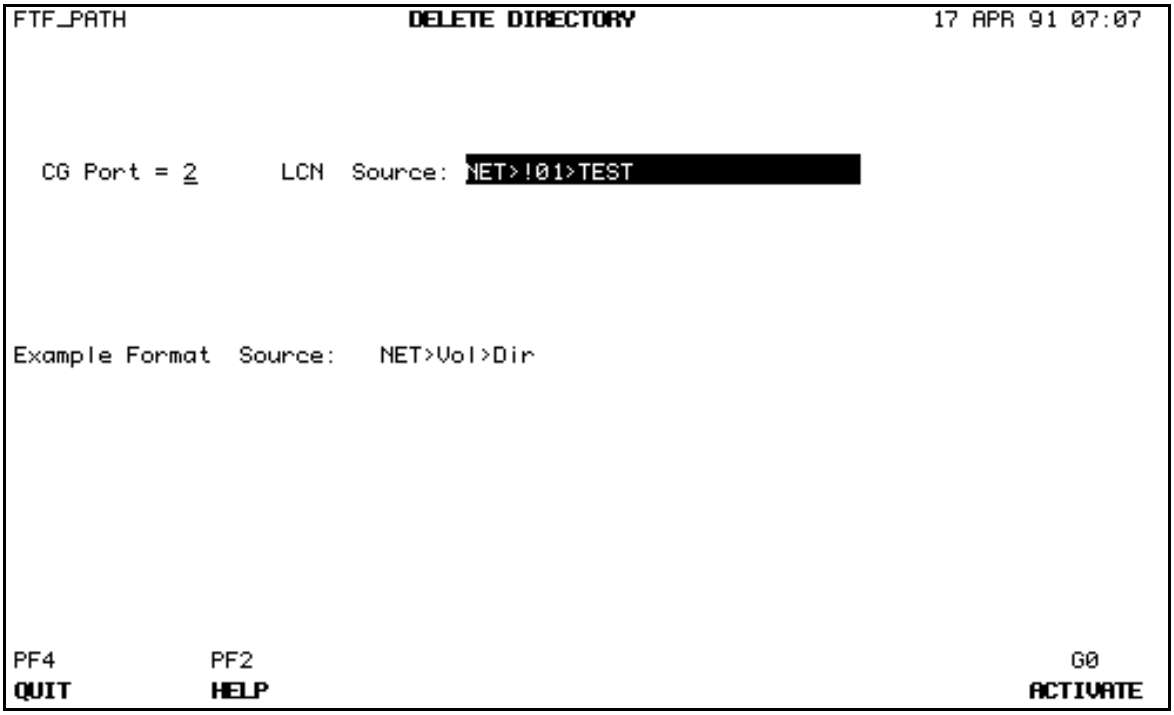
PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Starts the CREATE DIRECTORY process. Error and Information messages will be displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager also return a secondary error code that identifies the specific error (see Appendix A.4 for translation). The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>, will clear the error condition and allow for further processing.

7.3.12 Delete Directory

The DD menu option provides a screen to delete a directory under a volume on the History Module. No wildcard characters or dataout options are applicable.



7.3.12.1 Delete Directory Fields

The DELETE DIRECTORY fields are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• LCN Source—LCN pathname of the directory to be deleted. The device may be specified either by node number (PN:nn>) or as NET>. Example format: `NET>VOL> DIR` (Note the space delimiter before the directory name.)

7.3.12.2 Delete Directory Function Keys

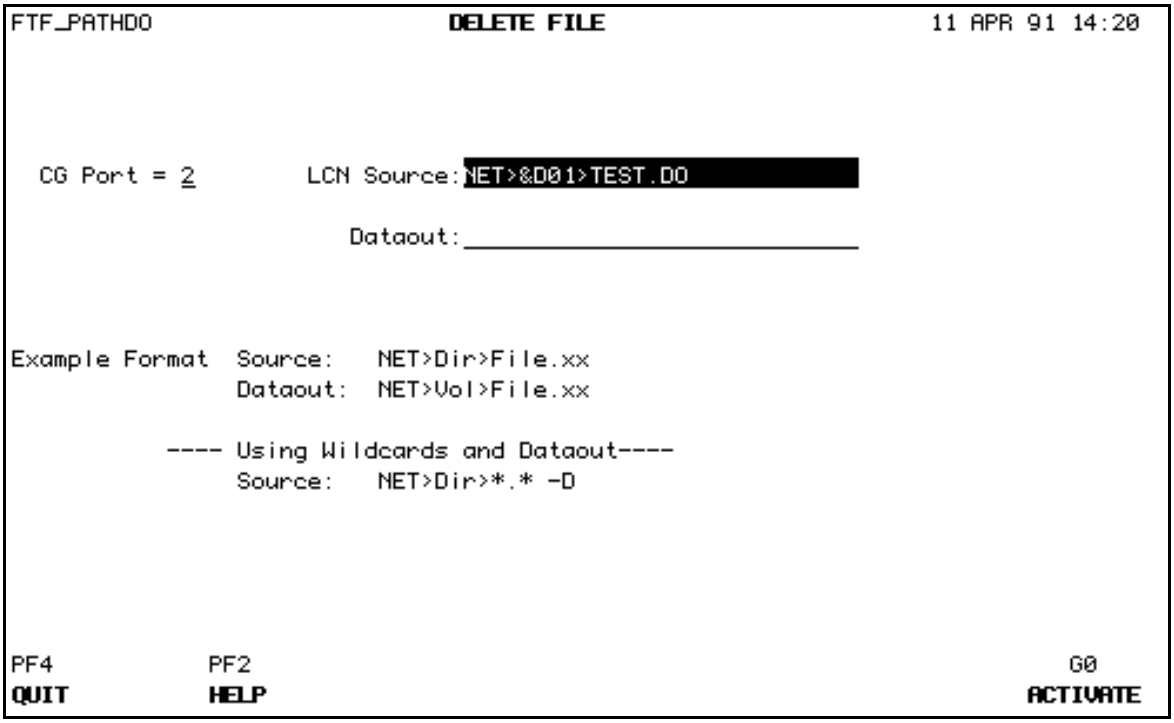
PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Starts the DELETE DIRECTORY process. Error and Information messages will be displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager also return a secondary error code that identifies the specific error (see Appendix A.4 for translation). The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>, will clear the error condition and allow for further processing.

7.3.13 Delete File

The DL menu option provides a screen to delete a file from the specified volume on the History Module. Wildcards permit deleting multiple files. Optionally, the actions can be journalized to a dataout file.



7.3.13.1 Delete File Fields

The DELETE FILE fields are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• LCN Source—LCN pathname of the file to be copied. The device may be specified either by node number (PN:nn>) or as NET>. If the LCN includes a Network Gateway, remote nodes can be addressed by including a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname. Example format: NET>&D01>FILENAME.DO. Wildcards (*) are permitted for the filename and/or extension.

• Dataout—LCN pathname of dataout file to be used as a journal/cataloging file using form: NET>&dir>filename.DO if the " -D" option is specified on the end of the LCN Source pathname. **Failure to specify the -D option will not produce an error message and the file deletes will not be journaled.** Using the -D option with a blank (unassigned) dataout will result in an error. The dataout assignment can be changed by typing in the desired pathname on this screen. If a dataout file is changed, the dataout assignment request will be executed before the file deletions.

7.3.13.2 Delete File Function Keys

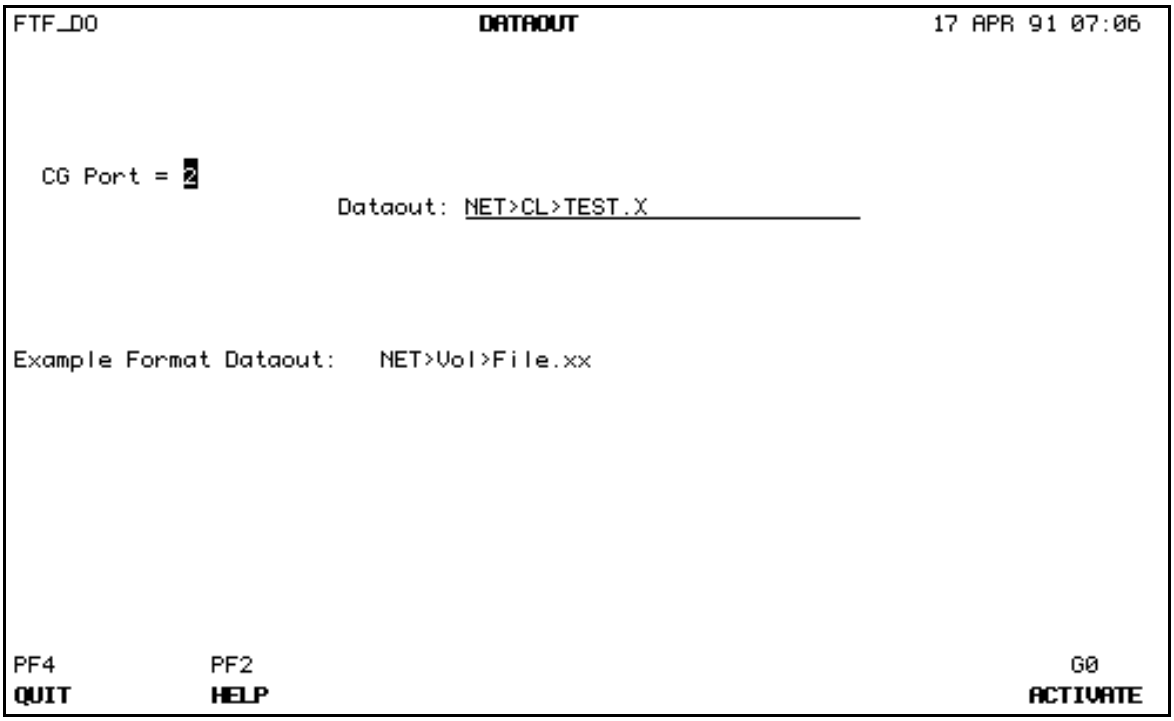
PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G-Ø/ACTIVATE—Start the DELETE FILE process using the CG Port and LCN Source parameters provided. If the dataout path is defined and the -D option is specified in the destination pathname, then journal the file deletes to the dataout file. Error and Information messages will be displayed at the bottom of the screen. Errors that are detected by either the LCN File or Utility Manager will also return a secondary error code (see Appendix A.4 for translation) that identifies the specific error. The secondary error message code will be highlighted and flashing. Pressing any field termination key, such as <Return>,will clear the error condition.

7.3.14 Dataout Status

The DO menu option provides a screen to view and change the current DATAOUT file assignment. This screen will display in the dataout field the current dataout assignment for the designated Computer Gateway. If a change is required, enter a new dataout pathname. Information messages will be received as the dataout status changes. If a reset of dataout is required, blank out the displayed data.



7.3.14.1 Dataout Status Fields

The DATAOUT STATUS fields are:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

• Dataout—LCN pathname of dataout file to be used as a journal/cataloging file using form: NET>&DIR>FILENAME.xx

If the LCN includes a Network Gateway, remote nodes can be addressed by including a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

7.3.14.2 Dataout Status Function Keys

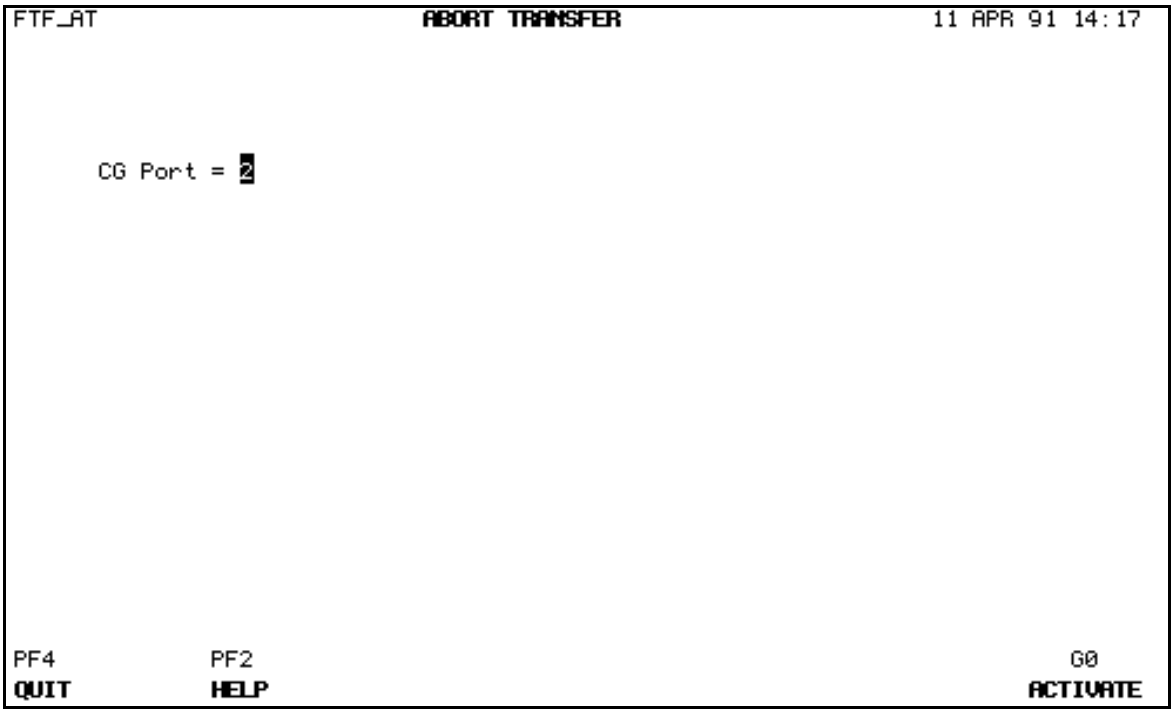
PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Sends the Dataout Request to the specified CG. Error and Information messages will be displayed at the bottom of the screen.

7.3.15 Abort Transfer

The AT menu option provides a screen to send an abort message to the Computer gateway. The current file transfer transaction will be terminated. The initiator of the aborted transaction will receive a CM50_FTF_ABORT error code. This screen will display on the 25th line the message "abort transfer request complete" even if no transaction was in progress at the time of the request.



7.3.15.1 Abort Transfer Fields

Only one field is used:

• CG Port—This field indicates the number of the CG Port (1- 4) to be used for the file transfer request.

7.3.15.2 Abort Transfer Function Keys

PF4/QUIT—Return to the File Transfer Menu.

PF2/HELP—Provides help information -- see Section 3.1.3.

G0/ACTIVATE—Sends the Abort Transfer Request to the specified CG. Error and Information messages will be displayed at the bottom of the screen.

DCL COMMAND INTERFACE
Section 8

This section explains how to invoke specific CM50S functions as DCL level commands.

8.1 USING THE COMMAND INTERPRETER

The DCL Interface to CM50S provides an easy to use, low volume, method for performing most of the standard CM50S functions. This interface follows the standard VMS DCL syntax rules and conventions.

8.1.1 Incorporating CM50S Commands into DCL

The CM50S ACP and DDT commands may be incorporated into DCL on either a system wide basis, or only for selected users.

8.1.1.1 System Wide Implementation

The CM50S ACP and DDT commands can be inserted into the system's DCL Table (by a user with full VMS privileges) by issuing the commands:

```
SET COMMAND/TABLE=SYS$LIBRARY:DCLTABLES -
/OUTPUT=SYS$LIBRARY:DCLTABLES CM50$LIB:ACP_COMMAND
```

and

```
SET COMMAND/TABLE=SYS$LIBRARY:DCLTABLES -
/OUTPUT=SYS$LIBRARY:DCLTABLES CM50$LIB:DDT_COMMAND
```

This will make the CM50S commands act like normal DCL commands, available to all users on the system. (Note: TheVAX system must be rebooted to activate the new DCL table; also these commands will need to be reissued whenever a new release of either VMS or CM50S is installed on the system.)

These commands can be removed from the system-wide DCL Table (by a user with full VMS privileges) by issuing the commands:

```
SET COMMAND/TABLE=SYS$LIBRARY:DCLTABLES/DELETE=ACP
SET COMMAND/TABLE=SYS$LIBRARY:DCLTABLES/DELETE=DDT
```

8.1.1.2 Providing Commands to Selected Users

The CM50S command interpreters are stored in the CM50$SUPPORT: logical directory. (Note: VMS ACLs [Access Control Lists] can be applied to this directory to limit the availability of the CM50S tools that it contains.) The DCL command:

```
SET COMMAND CM50$LIB:ACP_COMMAND, -
            CM50$LIB:DDT_COMMAND
```

adds the CM50S commands to the user's process table for the current terminal session. Once these special commands have been placed in the process table, they are accessible through the standard VMS Command Interpreter.

This command should be included in the LOGIN.COM file of users who will regularly exercise these commands.

8.1.1.3 Translating Error Codes

All CM50S commands return status codes in the standard VMS format. In order for a user to see the text message associated with these codes, his process must have issued the command:

```
SET MESSAGE CM50$LIB:CM50_ERROR_MSG
```

This command should be included in the LOGIN.COM file of those who will regularly use CM50S.

8.1.2 Options and Qualifiers

All CM50 command functions are grouped under two general commands: ACP and DDT. The specific functions to be performed are specified using Options and Qualifiers. The syntax for these commands matches the standard VMS DCL formatting rules.

The Options are positional and must be separated by one of more blank spaces. The first option specifies the function to be executed, and is always required. The remaining options depend on the function and must appear in the order shown.

The Qualifiers provide the ability to override the default settings for a function. Qualifiers are not positional, they may be placed at the end of any option preceded by a slash (/). All qualifiers are optional.

Both Option and Qualifier keywords may be abbreviated to 4 characters (or less if the result is still unique within the command). These keywords are not case sensitive.

8.1.3 Help and Error Handling

Whenever a DCL command function failures, a standard VMS error message is displayed pinpointing the cause. The return_status code is also stored in the symbol $STATUS so it can be tested using the DCL lexical functions.

On-line help is available within each command. The request for help is made by:

```
        ACP HELP topic
or      DDT HELP topic
```

The keyword HELP can be abbreviated to the single letter 'H'. The topics are the names of the functions that can be executed. If no topic is specified, a general help screen listing the available topics is displayed.

8.2 RETRIEVING LCN DATA

These commands retrieve current data about a tagname (point.parameter) from the LCN.

8.2.1 Viewing LCN Values

This command retrieves the value for a specified tag name from the LCN and displays it to the user. If the requested tagname is accessible, the output is a single line on the user's SYS$OUTPUT (normally the terminal screen) that reports its data type (Real, ASCII, etc.) and current value.

8.2.1.1 Example Commands to View an LCN Value

Minimum form:

```
ACP VAL tagname
```

Complete form:

```
ACP VALUE/CG=n/TYPE=nn tagname
```

8.2.1.2 Options for View Value

VALUE — Keyword identifying the view value function.

tagname — A string of up to 40 characters that identifies the LCN value to be retrieved. If the tagname string includes any embedded spaces, it must be enclosed in quotes. A tag name string is formatted as: *point.param[param_ix]*

where *point* is the ASCII name of the LCN point, optionally preceded by a pinid (1 or 2 characters and a backslash delimiter) for Network Gateway routing. It must be separated from the parameter name by a period (.).

param is the ASCII name of the LCN parameter whose value is to be displayed.

[param_ix] is the optional index to an element within an array of parameters, used only with arrays. When used, it must be an integer enclosed in square brackets ([ ]). If an array index is not specified, it is assumed to be zero.

8.2.1.3 Qualifiers for View Value

/CG=n—*(CG_port_number)* The port number (n = 1 to 4) of the CG where the ACIDP is resident. Defaults to 1.

/TYPE=nn—Specifies the data type to be returned. Defaults to 1. If the actual data type is not compatible with the requested type, the actual data type is reported with the value. If an array type is specified, the correct size of the array is returned along with the value. The legal data types are:

1 = Real	7 = Array of Reals
2 = Integer	8 = Array of Integers
3 = ASCII (24-char)	
4 = Enumeration	9 = Array of Enumerations
5 = Ordinal	10 = Array of Ordinals
13 = Internal ID	14 = Array of Internal IDs
15 = External ID	16 = Array of External IDs
17 = Time	18 = Array of Times
19 = String (24-char)	20 = Array of Strings

8.2.2 History Module Collection Rate

This command displays the frequency with which Continuous History is being collected for a specified tag name from the LCN. If the requested tagname is accessible, the output is a single line on the user's SYS$OUTPUT (normally the terminal screen).

8.2.2.1 Example Commands to Retrieve Collection Rate

Minimum form:

```
ACP RATE tagname
```

Complete form:

```
ACP RATE/CG=n tagname
```

8.2.2.2 Options for Collection Rate

RATE — Keyword identifying the view value function.

tagname — A string of up to 40 characters that identifies the LCN entity to be retrieved. If the tagname string includes any embedded spaces, it must be enclosed in quotes. A tag name string is formatted as: *point.param[param_ix]*

where *point.* is the ASCII name of the LCN point,. It must be separated from the parameter name by a period (.).
param is the ASCII name of the LCN parameter whose value is to be displayed.
[param_ix] is the optional index to an element within an array of parameters, used only with arrays. When used, it must be an integer enclosed in square brackets ([ ]). If an array index is not specified, it is assumed to be zero.t

8.2.2.3 Qualifiers for Collection Rate

/CG=n—*(CG_port_number)* The port number (n = 1 to 4) of the CG where the ACIDP is resident. Defaults to 1.

8.3 MANIPULATING ACPS

These commands are functionally equivalent to the ACP Operations described in Chapter 5 of this manual.

8.3.1 Activate an ACP

This command causes an ACP to be triggered. By default, nothing (except the next DCL prompt) is displayed upon successful completion.

8.3.1.1 Example Commands to Activate an ACP

Minimum form:

```
ACP ACTI acpname
```

Complete form:

```
ACP ACTIVATE/REMOTE acpname
```

8.3.1.2 Options for ACP Activate

ACTIVATE — Keyword identifying the activation function.

acpname — The name of the ACP to be activated.

8.3.1.3 Qualifiers for ACP Activate

/REMOTE — (Default) The ACP to be activated as a detached process.
OR
/INTERACTIVE — The ACP to be run interactively within the current process.

8.3.2 Deactivate an ACP

This command causes a running ACP to be terminated. By default, nothing (except the next DCL prompt) is displayed upon successful completion.

Note that deactivating an ACP issues a VMS STOP process command that kills the ACP without waiting for completion of any pending LCN call. If the ACP is accessing a DDT when it is deactivated, that DDT may be left locked.

To unlock a DDT: RUN CM50$SUPPORT:CM50_CLEAR_DDTUSE. This utility prompts for the DDT name, then informs the user of its success.

8.3.2.1 Example Commands to Deactivate an ACP

Minimum form:

```
ACP DEAC acpname
```

Complete form:

```
ACP DEACTIVATE/ABORT acpname
```

8.3.2.2 Options for ACP Deactivate

DEACTIVATE — Keyword identifying the deactivate function.

acpname — The name of the ACP to be deactivated.

8.3.2.3 Qualifiers for ACP Deactivate

/OFF — (Default) Set the ACP status to OFF, allowing it to be re-started from the LCN.
OR
/ABORT — Sets the ACP execution status to ABORT, preventing it from being restarted from the LCN

8.3.3 Install an ACP

This command causes a program/image to be installed as an ACP. Nothing (except the next DCL prompt) is displayed upon successful completion. Note: This command always uses the default quotas. If the ACP requires special privileges and/or quotas, the installation cannot be done at the DCL level.

8.3.3.1 Example Commands to Install an ACP

Minimum form:

```
ACP INST acpname
```

Complete form:

```
ACP INSTALL/PATH=file/TEST/INPUT=inputfile/OUTPUT=outputfile: -
      PROCESS_NAME=name/UIC=uic/PRIORITY=nn -
      /SYSERROR=errorfile  acpname
```

8.3.3.2 Options for ACP Installation

INSTALL — Keyword identifying the installation function.

acpname — The name (up to 12 characters long) of the ACP to be installed.

8.3.3.3 Qualifiers for ACP Installation

/PATH —*(executable_file)* The pathname of the executable image to be run when the ACP is triggered. If a simple file name (no directory specification) is given, the default directory of CM50$ACP is assumed. By default the executable_file has the same name as the ACP. If no extension is specified, the extension defaults to .EXE.

/NORMAL — (Default) The ACP will run normally through ACIDP triggers.
OR
/RESTRICTED — The ACP will not store any values and initiation by the LCN scheduler will be suppressed.
OR
/TEST — The ACP will not store any values and DDT Get operations will use the Test values instead of actual LCN values.

/INPUT—*(inputfile)* Allows a specified pathname of a file to be used as SYS$INPUT for the ACP. If given, inputfile should include the directory specification. ".DAT" is the default extension.

/OUTPUT—*(outputfile)* Allows a specified pathname of a file to be used as SYS$OUTPUT for the ACP. If given, outputfile should include the directory specification. ".DAT" is the default extension. To suppress the creation of an output file, specify the null device (NL:).

/PROCESS_NAME—*(proc_name)* Allows the name (up to 15 characters) of the detached process executing the ACP to be specified. By default, the ACP name is used as the process name.

/UIC—*(UICname)* Allows a UIC name (of up to 12 characters) to be specified for running the ACP. By default, ACPs run under the same UIC as the initiating process.

/SYSERROR—*(errorfile)* Allows a specified pathname of a file to be used as SYS$ERROR when the ACP is executed remotely. If specified, errorfile should include the directory specification. ".DAT" is the default extension. To suppress the creation of an error file, specify the null device (NL:).

/PRIORITY—*(integer)* Allows the priority of the ACP when executed remotely to be set between 1 and 30. The default priority is 4.

8.3.4 Uninstall an ACP

This command causes an program to be removed from the ACP status table. Nothing (except the next DCL prompt) is displayed upon successful completion. If the ACP is connected to an ACIDP, it will be disconnected as part of this command. Note that the disconnect will fail if the ACP is in RESTRICTED mode.

8.3.4.1 Example Commands to Uninstall an ACP

Minimum form:

```
ACP UNIN acpname
```

Complete form:

```
ACP UNINSTALL acpname
```

8.3.4.2 Options for ACP Uninstall

UNINSTALL — Keyword identifying the uninstall function.

acpname — The name of the ACP to be uninstalled.

8.3.4.3 Qualifiers for ACP Uninstall

There are no qualifiers for this command.

8.3.5 Connect an ACP to an ACIDP

This command causes an ACP to be connected to an ACIDP. Nothing (except the next DCL prompt) is displayed upon successful completion.

8.3.5.1 Example Commands to Connect an ACP

Minimum form:

```
ACP CONN acpname acidp
```

Complete form:

```
ACP CONNECT/CG=n acpname acidp
```

8.3.5.2 Options for ACP Connect

CONNECT — Keyword identifying the connect function.

acpname — The name of the ACP to be connected.

acidp — The name of the ACIDP to be connected to the ACP.

8.3.5.3 Qualifiers for ACP Connect

/CG=n—*(CG_port_number)* The number of the CG (n = 1 to 4) where the ACIDP is resident. Defaults to 1.

8.3.6 Disconnect an ACP

This command causes an ACP to be disconnected from its ACIDP. Nothing (except the next DCL prompt) is displayed upon successful completion. Note that if the ACP is in RESTRICTED mode, its installation mode must be change to NORMAL or TEST before it can be disconnected.

8.3.6.1 Example Commands to Disconnect an ACP

Minimum form:

```
ACP DISC acpname
```

Complete form:

```
ACP DISCONNECT acpname
```

8.3.6.2 Options for ACP Disconnect

DISCONNECT — Keyword identifying the disconnect function.

acpname — The name of the ACP to be disconnected.

8.3.6.3 Qualifiers for ACP Disconnect

There are no qualifiers for this command.

8.3.7 Change Program Mode of an ACP

This command changes the installation mode of an ACP. Nothing (except the next DCL prompt) is displayed upon successful completion.

8.3.7.1 Example Commands to Change Installation Mode

Minimum form:

```
ACP CHAN acpname
```

Complete form:

```
ACP CHANGEMODE/RESTRICTED acpname
```

8.3.7.2 Options for Change Mode

CHANGEMODE — Keyword identifying the change mode function.

acpname — The name of the ACP to be affected.

8.3.7.3 Qualifiers for Change Mode

/NORMAL — (Default) The ACP will run normally through ACIDP triggers.
OR
/RESTRICTED — The ACP will not store any values and initiation by the LCN scheduler will be suppressed.
OR
/TEST — The ACP will not store any values and DDT Get operations will use the Test values instead of actual LCN values.

8.3.8 Display Status of an ACP

This command displays the current status of an ACP.

8.3.8.1 Example Commands to Display the Status of an ACP

Minimum form:

```
ACP SUM acpname
```

Complete form:

```
ACP SUMMARY acpname
```

8.3.8.2 Options for Display ACP Status

SUMMARY — Keyword identifying the display summary function.

acpname — The name of the ACP whose status is to be displayed.

8.3.8.3 Qualifiers for Display ACP Status

There are no qualifiers for this command.

8.3.9 Display List of ACPs

This command displays summary information on all installed ACPs.

8.3.9.1 Example Commands to List ACPs

Minimum form:

```
ACP LIS
```

Complete form:

```
ACP LIST/PAGE/BEGIN=nn/END=nn/NOLIST
```

8.3.9.2 Options for ACP List

LIST — Keyword identifying the list function.

8.3.9.3 Qualifiers for ACP List

/PAGE — The output is displayed one screen at a time. At the end of each page, the user must press <*Return*> to view the next screen. Typing: *Q*<*Return*>. Without this qualifier, the entire list will be displayed (scrolling as needed).

/BEGIN—*(starting_index)* The index number of the first ACP entry to be displayed. Default = 1.

/END—*(ending_index)* The index number of the last ACP to be displayed. Defaults to the end of the ACP table.

/NOLIST — The output is limited to a single line giving the current count of installed ACPs.

8.4 MANIPULATING DDTs

These commands are functionally equivalent to the DDT Operations described in Section 6 of this manual.

8.4.1 Build a DDT

This command causes a DDT to be built from a source text file. Nothing (except the next DCL prompt) is displayed upon successful completion. Note that rebuilding a CG-resident DDT breaks any existing ACIDP connection. Thus, if prefetching is desired, the DDT BUILD/REBUILD command should be followed by a DDT CONNECT command (see section 8.4.5).

8.4.1.1 Example Commands to Build a DDT

Minimum form:

```
DDT BUIL ddtname
```

Complete form:

```
DDT BUILD/SOURCE=sourcefile/DESCR=description/CG=n  -
    /RESIDENT/REBUILD/VT/LIST_ERRORS                -
    /NOERROR_FILE ddtname
```

8.4.1.2 Options for DDT Build

BUILD — Keyword identifying the build function.

ddtname — The name (up to 9 characters long) to be assigned to the DDT.

8.4.1.3 Qualifiers for DDT Build

/SOURCE_PATH—*(sourcefile)* The pathname of the source file for the DDT. If no directory is specified, the user's current default directory will be assumed. if no extension is given, an extension of ".DDT" will be assumed. By default, the sourcefile name is the same as the ddtname.

/DESCRIPTION—*("text ")* — A description of up to 36 characters to be associated with the DDT. If the description contains any embedded spaces, it must be enclosed in quotes.

/CG—*(CG_port_number)* The CG (n = 1 to 4) through which the data is to be transferred. Default = 1.

/NORESIDENT — (Default) The DDT to be maintained in the VAX and transferred to the CG at run time.
OR
/RESIDENT — The DDT to be installed as CG-resident.

/NOREBUILD — (Default) This is a new DDT; it will not overlay an existing DDT.
OR
/REBUILD — The existing DDT is to be replaced using the current source file.

/NOVT — (Default) The values transferred at run time are not recorded on disk.
OR
/VT — A Values_Table file is to be generated, recording the most recent values transferred.
(Note: This option reduces run-time throughput.)

/NOLIST — (Default) The source error file is not automatically displayed.
OR
/LIST_ERRORS — The source error file is displayed.

/ERROR_FILE — (Default) Errors in the source file are recorded in an error file (.ER) .
OR
/NOERROR_FILE — Errors in the source file are not not stored in a file.

8.4.2 Delete a DDT

This command deletes a DDT (except for its source file). Nothing (except the next DCL prompt) is displayed upon successful completion.

8.4.2.1 Example Commands to Delete a DDT

Minimum form:

```
DDT DELE ddtname
```

Complete form:

```
DDT DELETE ddtname
```

8.4.2.2 Options for DDT Delete

DELETE — Keyword identifying the Delete function.

ddtname — The name of the DDT to be deleted.

8.4.2.3 Qualifiers for DDT Delete

There are no qualifiers for this command.

8.4.3 Install a DDT as CG Resident

This command causes a DDT to be installed as CG resident. Nothing (except the next DCL prompt) is displayed upon successful completion.

8.4.3.1 Example Commands to Install a Resident DDT

Minimum form:

```
DDT INST ddtname
```

Complete form:

```
DDT INSTALL ddtname
```

8.4.3.2 Options for DDT Install

INSTALL — Keyword identifying the install function.

ddtname — The name of the DDT to be installed.

8.4.3.3 Qualifiers for DDT Install

There are no qualifiers for this command.

8.4.4 Remove a DDT from the CG

This command causes a resident DDT to be removed from its CG. If the DDT is connected to an ACIDP, that connection is also removed. Nothing (except the next DCL prompt) is displayed upon successful completion.

8.4.4.1 Example Commands to Remove a DDT

Minimum form:

```
DDT UNIN ddtname
```

Complete form:

```
DDT UNINSTALL ddtname
```

8.4.4.2 Options for DDT Uninstall

UNINSTALL — Keyword identifying the removal function.

ddtname — The name of the DDT to be removed from the CG.

8.4.4.3 Qualifiers for DDT Uninstall

There are no qualifiers for this command.

8.4.5 Connect a DDT to an ACIDP

This command causes a DDT to be built connected to an ACIDP, allowing its data to be prefetched. Nothing (except the next DCL prompt) is displayed upon successful completion.

8.4.5.1 Example Commands to Connect a DDT to an ACP

Minimum form:

```
DDT CONN ddtname acpname
   OR
DDT CONN/ACID=acidpname  ddtname
```

Complete form:

```
DDT CONNECT/ACIDP=acidpname/NOSCHEDULE/NODEMAND/NOPPS  -
     ddtname acpname
```

8.4.5.2 Options for DDT Connect

CONNECT — Keyword identifying the connect function.

ddtname — The name of the DDT to be connected for prefetch.

acpname — (Optional) The name of the ACP that will use the prefetched data. This ACP must already be connected to an ACIDP. If the ACP name is not given, then the /ACIDP qualifier must be used to specify the name of the ACIDP for the connection.

8.4.5.3 Qualifiers for DDT Connect

/ACIDP—*(acidp_name*) The name of the ACIDP for the connection. This ACIDP must already be connected to an ACP.

/SCHEDULE — (Default) The data will be prefetched when the ACP is triggered by the LCN Scheduler.
OR
/NOSCHEDULE — The data will be fetched only when requested if the ACP is triggered by the LCN Scheduler.

/DEMAND — (Default) The data will be prefetched when the ACP is triggered by Operator Demand.
OR
/NODEMAND — The data will be fetched only when requested if the ACP is triggered by Operator Demand.

/PPS — (Default) The data will be prefetched when the ACP is initiated by an LCN Event or program.
OR
/NOPPS — The data will be fetched only when requested if the ACP is initiated by an LCN Event or program.

8.4.6 Disconnect a DDT from an ACIDP

This command disconnects a DDT from its ACIDP. Nothing (except the next DCL prompt) is displayed upon successful completion.

8.4.6.1 Example Commands to Disconnect a DDT

Minimum form:

```
DDT DISC ddtname
 or
DDT DISC/ACP=acpname
 or
DDT DISC/ACIDP=acidname
```

Complete form:

```
DDT DISCONNECT/ACIDP=acidpname/ACP=acpname ddtname
```

8.4.6.2 Options for DDT Disconnect

DISCONNECT — Keyword identifying the disconnect function.

ddtname — (Optional) The name of the DDT to be disconnected. If not specified, then either the /ACIDP or /ACP qualifier must be used to identify the connection that is to be broken.

8.4.6.3 Qualifiers for DDT DISCONNECT

/ACIDP—*(acidp_name)* — The name of the connected ACIDP.

/ACP—*(acp_name)* — The name of the connected ACP.

8.4.7 Modify the Prefetch Triggers for a DDT

This command changes the triggers that cause data for a DDT to be prefetched. Nothing (except the next DCL prompt) is displayed upon successful completion.

8.4.7.1 Example Commands to Modify Prefetch Triggers

Minimum form:

```
DDT MOD ddtname
 or
DDT MOD/ACP=acpname
 or
DDT MOD/ACIDP=acidpname
```

Complete form:

```
DDT MODTRIGGERS/ACIDP=acidpname/ACP=acp_name  -
     /NOSCHEDULE/NODEMAND/NOPPS ddtname
```

8.4.7.2 Options for Modify Triggers

MODTRIGGERS — Keyword identifying the modify triggers function.

ddtname — (Optional) The name of the DDT to be disconnected. If not specified, then either the /ACIDP or /ACP qualifier must be used to identify the connection that is to be modified.

8.4.7.3 Qualifiers for Modify Triggers

/ACIDP—*(acidp_name)* The name of the connected ACIDP.

/ACP—*(acp_name)* The name of the connected ACP.

/SCHEDULE — (Default) The data will be prefetched when the ACP is triggered by the LCN Scheduler.
OR
/NOSCHEDULE — The data will be fetched only when requested if the ACP is triggered by the LCN Scheduler.

/DEMAND — (Default) The data will be prefetched when the ACP is triggered by Operator Demand.
OR
/NODEMAND — The data will be fetched only when requested if the ACP is triggered by Operator Demand.

/PPS — (Default) The data will be prefetched if the ACP is initiated by an LCN Event or program.
OR
/NOPPS — The data will be fetched only when requested if the ACP is initiated by an LCN Event or program.

8.4.8 Display Summary Information for a DDT

This command displays summary information about a DDT. The output shows the description and status (but not the component points) of the named DDT.

8.4.8.1 Example Commands to Display a DDT Summary

Minimum form:

```
DDT SUM ddtname
```

Complete form:

```
DDT SUMMARY ddtname
```

8.4.8.2 Options for DDT Summary

SUMMARY — Keyword identifying the summary display function.

ddtname — The name of the DDT to be displayed.

8.4.8.3 Qualifiers for DDT Summary

There are no qualifiers for this command.

8.4.9 Display Detailed Information for a DDT

This command displays a DDT including all of its points.

8.4.9.1 Example Commands to Display a DDT's Details

Minimum form:

```
DDT DETA ddtname
```

Complete form:

```
DDT DETAIL/PAGE ddtname
```

8.4.9.2 Options for DDT Detail

DETAIL — Keyword identifying the detailed display function.

ddtname — The name of the DDT to be displayed.

8.4.9.3 Qualifiers for DDT Detail

/PAGE — The output is displayed one screen at a time. At the end of each page, the user must press either <*Return*> to view the next screen, or *Q*<*Return*> to terminate the command with displaying the rest of the list. If the /PAGE qualifier is not used, the entire list of points is displayed at once (scrolling as needed).

8.4.10 Display List of DDTs

This command displays a list of DDTs, with a summary entry for each DDT and a count of the current number of DDTs.

8.4.10.1 Example Commands to List DDT

Minimum form:

```
DDT LIS
```

Complete form:

```
DDT LIST/BEGIN=nn/END=nn/NOLIST/PAGE
```

8.4.10.2 Options for DDT List

LIST — Keyword identifying the list function.

8.4.10.3 Qualifiers for DDT List

/PAGE —The output is displayed one screen at a time. At the end of each page, the user must press either <*Return*> to view the next screen, or *Q*<*Return*> to terminate the command with displaying the rest of the list. If the /PAGE qualifier is not used, the entire list of points is displayed at once (scrolling as needed).

/BEGIN—*(starting_index)* Index number of the first DDT to be displayed. Defaults to 1.

/END—*(ending_index)* Index number of the last DDT to be displayed. Defaults to the end of the list.

/NOLIST — Limits the output to a single line that reports the total number of DDTs.

8.5 TRANSFERRING LCN FILES

These commands are functionally equivalent to the LCN File Transfer Operations described in Section 7.3 of this manual.

The Dataout facility allows the user, when requesting the execution of specific file transfer transactions, to place relevant data in the dataout or catalog file. This dataout file is a shared file for all users of file transfer on the LCN. For example, user "Jones" executes a D_ATTRIBUTES command, the results of which are placed into the current dataout file. User "Smith" then requests a D_VOLUME command. These results also are placed into the same (current) dataout file. The D_ATTRIBUTES and D_VOLUME commands are the only file transfer operations that require a dataout file. Other file transfer transactions treat dataout as an option.

8.5.1 LCN File Read

This command will transfer a single file from an LCN NET volume to CM50S. Wildcard transfers of files are not supported. This command also creates an "LCN ATTRIBUTES" file for every LCN file that is transferred. Multiple transfers of the same LCN file, within the same VMS directory, are not allowed. The version number of the attributes file should remain 1. The attributes file contains the necessary information for transferring the file back to the LCN. This includes block and record size, file size and type, and file descriptor data.

8.5.1.1 Example Command to Read an LCN Resident File.

Minimum form:

```
FTF READ lcnfile vaxfile
```

Complete form:

```
FTF READ/CG=n  lcnfile vaxfile  /ACIDP=acidp_name
```

8.5.1.2 Options for File Read

READ—Keyword identifying the File Read function.

lcnfile—Pathname (up to 28 characters) of the LCN file to be read. Use the form: PN:nn>&DIR>FILENAME.xx or NET>&DIR>FILENAME.xx. If the LCN includes a Network Gateway, remote nodes can be addressed by adding a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

vaxfile—VMS file pathname (up to 80 characters) where the file will be stored in the VAX. If no directory is specified, the user's current default directory will be used. If no extension is specified, the VMS default of .DAT will be used. The LCN attributes file will use the following naming convention: the filename suffix or extension will be preceded by an under-bar character, followed by a period "LA" extension. For example; the LCN filename of FORMULAE.CL would have an attribute file of FORMULAE_CL.LA.

8.5.1.3 Qualifiers for File Read

/CG=*(cg_port_number)*—Specifies which Computer Gateway (1-4) to use for access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.2 LCN File Write

This command will transfer a single file from CM50S to LCN NET volume. This command requires the LCN ATTRIBUTES file for every LCN file that is transferred. Multiple copies of an LCN file within the same VMS directory are allowed. These files would have been created by modifying the original LCN file which was transferred as version 1. The version number of the attributes file should remain 1. The attributes file should not be modified.

8.5.2.1 Example Command to Write an LCN Resident File.

Minimum form:

```
FTF WRIT  vaxfile lcnfile
```

Complete form:

```
FTF WRITE/CG=n/REPLACE  vaxfile lcnfile  -
     /ACIDP=acidp_name
```

8.5.2.2 Options for File Write

WRITE—Keyword identifying the file write function.

vaxfile—VMS file pathname (up to 80 characters) of the VAX resident file to transfer to the LCN NET volume. A valid attributes file must also reside in the same directory. The LCN attributes file uses the following naming convention: the filename suffix or extension will be preceded by an under-bar character, followed by a period "LA" extension. For example, the LCN filename of FORMULAE.CL would have an attribute file of FORMULAE_CL.LA.

lcnfile—LCN file pathname (up to 28 characters) using the form: PN:nn>&DIR>FILENAME.xx or NET>&DIR>FILENAME.xx

If the LCN includes a Network Gateway, remote nodes can be addressed by adding a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

8.5.2.3 Qualifiers for File Write

/REPLACE—If the named LCN file already exists, replace it.
OR
/ABORT—(default) Abort the transfer if the named LCN file already exists. Do not overwrite any existing file.

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.3 List LCN Filenames

This command will list up to 1180 file names and extensions from a LCN NET volume. If the number of files exceeds the buffer capacity of 1180, then multiple requests by directory, file type extension, or filename syntax must be executed. Wildcards are permitted. The list format is one filename.extension per line.

8.5.3.1 Example Command to List Filenames and Extensions.

Minimum form:

```
FTF FILE  lcnfile
```

Complete form:

```
FTF FILE_LIST/CG=n/OUTPUT=comfil lcnfile
/ACIDP=acidp_name
```

8.5.3.2 Options for List Filenames

FILE_LIST—Keyword identifying the file list function.

lcnfile—LCN file pathname (up to 28 characters), usually with a wildcard in one of the forms:

NET>&DIR>FILENAME.*
NET>&DIR>*.xx
PN:nn>&DIR>*.*

If the LCN includes a Network Gateway, remote nodes can be addressed by adding a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

8.5.3.3 Qualifiers for List Filenames

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/OUTPUT=*(comfile*)—Specifies the VAX pathname for a command file. If this qualifier is used, the list of file names is formatted into a command procedure for reading all of the files into the current default directory in the VAX. Thus, the simplest way to archive all of the files from a directory on the LCN to a directory on the VAX is with a pair of commands like these:

```
$FTF FILE/OUTPUT=ARCHIV.com NET>CL>*.*
$@ARCHIV
```

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.4 List LCN File Attributes

This command will list the file attributes for a specific LCN file. Wildcard characters and dataout options are not permitted. File attributes are:

- LCN file type—contiguous or linked
- LCN file protection
- Record size
- Block size
- LCN file configuration
- LCN file revision
- Directory timestamp (MM/DD/YY hh:mm)
- Logical number of blocks
- Logical number of records
- File Descriptor
- Starting Sector
- Ending Sector

8.5.4.1 Example Command to List File Attributes.

Minimum form:

```
FTF ATTR lcnfile
```

Complete form:

```
FTF ATTRIBUTES/CG=n  lcnfile  /ACIDP=acidp_name
```

8.5.4.2 Options for List File Attributes

ATTRIBUTES—Keyword identifying the file transfer function.

lcnfile—LCN file pathname (up to 28 characters) using the form:
PN:nn>VDIR>FILENAME.xx or NET>VDIR>FILENAME.xx.

If the LCN includes a Network Gateway, remote nodes can be addressed by adding a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

8.5.4.3 Qualifiers for List File Attributes

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.5 List File Attributes to Dataout

This file transfer command will list the LCN FILE ATTRIBUTES of 1 or more files into the current dataout file. Wildcards (*) may be used for the filename and/or extension. Two options allow selection of which attributes are listed: -FD lists file descriptors, -REC lists record & block attributes.

The dataout file need not have been previously established. The complete absence of a dataout specification will result in an error return. By using the CAT_FILE qualifier a dataout file may be specified at execution time. Failure to specify a dataout file either prior to command execution or by using the CAT_FILE qualifier will result in an error return.

8.5.5.1 Example Command to List File Attributes using Dataout.

Minimum form:

```
FTF D_AT  lcnfile
```

Complete form:

```
FTF D_ATTRIBUTES/CAT_FILE=dataout  "lcnfile -FD -REC" -
     /CG=n /ACIDP=acidp_name
```

8.5.5.2 Options for List Attributes to Dataout

D_ATTRIBUTES—Keyword identifying this listing function.

lcnfile—LCN file pathname (up to 28 characters) using the form:

PN:nn>&DIR>filename.xx	or
NET>&DIR>filename.xx.	or
NET>&DIR>*.xx	or
NET>&DIR>filename.*	if wildcards are used, or
"NET>&DIR>FILE.xx -FD -REC"	(with double quotes) if options are used.

8.5.5.3 Qualifiers for List Attributes to Dataout

/CAT_FILE=*(dataout)*—Dataout file pathname on the LCN (up to 28 characters) where the listing will be stored. Use the form NET>&DIR>FILENAME.xx.

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.6 List LCN Volumes

This command will list History Module volume and directory names as well as sector usage figures. Wildcards and dataout options are not permitted for this transaction type.

8.5.6.1 Example Command to List LCN Volumes by Device.

Minimum form:

```
FTF VOL lcnnode
```

Complete form:

```
FTF VOLUME/CG=n  lcnnode  /ACIDP=acidpname
```

8.5.6.2 Options for Listing LCN Volumes

VOLUME—Keyword identifying this listing function.

lcnnode—LCN node name using the form: PN:nn (where nn is the node number).

8.5.6.3 Qualifiers for Listing LCN Volumes

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.7 Listing LCN Volumes to Dataout

This file transfer command will list the LCN Volumes and Directories for all History modules on the NET or for a specific History Module. The dataout file need not have been previously established. The complete absence of a dataout specification will result in an error return. By using the CAT_FILE qualifier the dataout file may be specified at execution time.

8.5.7.1 Example Command to List Volumes to Dataout.

Minimum form:

```
FTF D_VO  lcnnode
```

Complete form:

```
FTF D_VOLUME/CG=n/CAT_FILE=dataout  lcnnode  -
     /ACIDP=acidpname
```

8.5.7.2 Options for Listing Volumes to Dataout

D_VOLUME—Keyword identifying this listing function.

lcnnode—LCN node name using the form: PN:nn (where nn is the node number) or NET

8.5.7.3 Qualifiers for Listing Volumes to Dataout

/CAT_FILE=*(dataout)*—Dataout file pathname on the LCN (up to 28 characters) where the listing will be stored. Use the form NET>&DIR>FILENAME.xx.

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.8 Copy LCN File

This file transfer command will copy a single file or all files from one NET volume to another Net volume. Wildcards are permitted; however, the destination suffix must always be the same as the source suffix.

The -D option is supported for journalizing all copies to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the copy function will not be executed.

8.5.8.1 Example Command to Copy Files.

Minimum form:

```
FTF COPY  lcnfile1  lcnfile2
```

Complete form:

```
FTF COPY/CG=n  lcnfile1 "lcnfile2 -D"  /ACIDP=acidpname
```

8.5.8.2 Options for File Copy

COPY—Keyword identifying file copy function.

lcnfile1—Source LCN file pathname (up to 28 characters) using the form:
PN:nn or NET>FILENAME.nn, or
NET>&DIR>*.* if using wildcards.

lcnfile2—Recipient LCN file pathname (up to 28 characters) using the form:
NET>VDIR>FILENAME. The file extension must remain the same and will be automatically retained. The form "NET>VDIR>FILENAME -D" would be used if a dataout journal is being used. (The double quotes are required.)

If the LCN includes a Network Gateway, remote nodes can be addressed by adding a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

8.5.8.3 Qualifiers for File Copy

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.9 Move LCN Files

This file transfer command will move a single file or all files from one directory to another directory within the same NET volume. Wildcards are permitted.

The -D option is supported for journalizing all moves to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the move function will not be executed.

8.5.9.1 Example Command to Move an LCN File

Minimum form:

```
FTF MOVE  lcnfile  lcndir
```

Complete form:

```
FTF MOVE/CG=n  lcnfile "lcndir -D"  /ACIDP=acidpname
```

8.5.9.2 Options for File Move

MOVE—Keyword identifying move file function.

lcnfile—LCN file pathname (up to 28 characters) using the form:
using the form NET>DIR1>FILENAME.nn, or
NET>DIR1>*.* if using wildcards.

lcndir—Name of the LCN directory that will receive the file.name. Only the directory name is required because the move must be within the same volume. Example: FTF MOVE NET>VDIR>FILENAME.xx DIR2. The form "DIR2 -D" in double quotes would be specified if a dataout journal is to be used.

8.5.9.3 Qualifiers for File Move

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.10 Rename LCN File

This file transfer command will rename a single file or all files on the History Module. Wildcards are permitted.

The -D option is supported for journalizing all renames to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the rename function will not be executed.

8.5.10.1 Example Command to Rename an LCN File

Minimum form:

```
FTF RENA  lcnfile1 lcnfile2
```

Complete form:

```
FTF RENAME/CG=n  lcnfile1 "lcnfile2 -D"
/ACIDP=acidpname
```

8.5.10.2 Options for File Rename

RENAME—keyword identifying file rename function.

lcnfile1—Old LCN file pathname (up to 28 characters) using the form: NET>DIR1>FILENAME.nn, or NET>DIR1>*.nn if using wildcards.

lcnfile2—New LCN filename (up to 28 characters). The Directory and extension will be automatically provided (The extension cannot be changed with this command.) The form: "FILENAME -D" may be used to journal the change to Dataout. The double quotes are required with the -D option.

8.5.10.3 Qualifiers for File Rename

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.11 Delete LCN File

This file transfer command will delete a single file or all files from the specified volume on the History Module. Wildcards are permitted. **Once deleted the file cannot be recovered.**

The -D option is supported for journalizing all deleted files to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the delete file function will not execute.

8.5.11.1 Example Command to Delete an LCN File

Minimum form:

```
FTF DELE  lcnfile
```

Complete form:

```
FTF DELETE/CG=n  "lcnfile -D"  /ACIDP=acidpname
```

8.5.11.2 Options for File Delete

DELETE—keyword identifying file delete function.

lcnfile—LCN file pathname (up to 28 characters) using the form:

NET>&DIR>FILENAME.nn,	or
NET>&DIR>*.*	if using wildcards, or
"NET>&DIR>*.* -D"	if using dataout.

If the LCN includes a Network Gateway, remote nodes can be addressed by adding a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

8.5.11.3 Qualifiers for File Delete

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.12 Directory Transactions

This file transfer command will create or delete a directory under a volume on the History Module. No wildcards characters or options are permitted.

8.5.12.1 Example Command to Create or Delete an LCN Directory

Minimum form:

```
FTF DIR "lcndir"
```

Complete forms:

```
FTF DIRECTORY/CREATE/CG=n "lcndir" /ACIDP=acidpname

FTF DIRECTORY/DELETE/CG=n "lcndir" /ACIDP=acidpname
```

8.5.12.2 Options for Directory Maintenance

DIRECTORY—Keyword identifying directory maintenance function.

> **lcndir—**LCN directory pathname using the form:"NET>VOL> DIR" (note the space delimiter before the directory name). Directory names are up to 4 characters, and **the double quotes are required**.

8.5.12.3 Qualifiers for Directory Maintenance

/DELETE—Delete the named LCN directory.
OR
/CREATE—(default) Create the named LCN directory.

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.13 Dataout Request

The dataout function allows the user, when requesting the execution of specific file transfer transactions, to place relevant data in the dataout or catalog file. This dataout file is a shared file by all concurrent users of file transfer. For example, user "Jones" requests a DATA_OUT_FILE command, the results of which are placed into the current dataout file. User "Smith" then executes a DATA_OUT_VOLUME command. These results also are placed into the same (current) dataout file. The dataout file may be transferred to the VAX host using the file transfer READ request. The DATA_OUT_REQUEST command is provided to enable or disable the file transfer dataout file definition.

8.5.13.1 Example of Dataout Request

Minimum form:

```
FTF DATA  lcnfile
```

Complete form:

```
FTF DATA_OUT_REQUEST/CG=n/ENABLE  lcnfile  -
     /ACIDP=acidpname
```

8.5.13.2 Options for Dataout Request

DATA_OUT_REQUEST—Keyword identifying the Dataout Request function.

lcnfile—LCN file pathname (up to 28 characters) for Dataout using the form:
NET>&DIR>FILENAME.nn.

If the LCN includes a Network Gateway, remote nodes can be addressed by adding a prefix—consisting of the one- or two-character "pinid" followed by a backslash—to the pathname.

8.5.13.3 Qualifiers for Dataout Request

/ENABLE—Causes the named LCN file to be opened as the recipient of Dataout cataloging for subsequent file transfer operations.
OR
/DISABLE—(Default) This LCN file name although required, is not verified against the current dataout assignment. The current dataout file will be unconditionally disabled.

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.14 Dataout Status

The D_STATUS command identifies the current dataout assignment on a CG.

8.5.14.1 Example Command to Report Dataout Status

Minimum form:

```
FTF D_ST   lcnfile
```

Complete form:

```
FTF D_STATUS/CG=n  lcnfile  /ACIDP=acidpname
```

8.5.14.2 Options for Dataout Status

D_STATUS—Keyword identifying the Dataout Status function.

lcnfile—LCN file pathname (up to 28 characters) of the Dataout file using the form: NET>&DIR>FILENAME.nn.

8.5.14.3 Qualifiers for Dataout Status

/CG=*(cg_port_number)*—Specifies which Computer gateway (1-4) to use to access to the LCN. Default value is 1.

/ACIDP=(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

8.5.15 Abort File Transfer

This command ABORT_TRANSFER will terminate the current transaction in progress. The initiator of the transaction in progress will receive a CM50_FTF_ABORT error return status. No error is reported to the requestor if there is not a current process to abort.

8.5.15.1 Example of Command to Abort a File Transfer

Minimum form:

```
FTF ABOR CGnbr
```

Complete form:

```
FTF ABORT_TRANSFER CGnbr  /ACIDP=acidpname
```

8.5.15.2 Options for Abort Transfer

ABORT_TRANSFER—Keyword identifying the file transfer abort function.

CGnbr—Number (1-4) specifying which Computer Gateway is to have its File Transfer aborted. No Default CG Port Number is provided.

8.5.15.3 Qualifiers for Abort Transfer

/**ACIDP=**(*acidp_name*)—Reserved for future security use (will specify an ACIDP that controls the LCN data access).

FORTRAN LANGUAGE CONSIDERATIONS
Section 9

This section discusses each of the program interfaces that provide necessary services that enable FORTRAN programs to communicate with other nodes on the TDC 3000 Local Control Network.

9.1 CM50S INCLUDE FILES

Each user interface routine has language-specific interfaces that are supported by include files that contain data declarations that match the argument names and symbolic constants used in the example calls in this section. **Any program that uses any of these interface routines should be compiled with the matching language-specific include files.**

9.1.1 Include Files for Data Transfer Functions

These include files generally are needed by Advanced Control Programs and Data Acquisition programs.

`CM50$LIB:CM50_INCLUDE.FOR`	Contains the declarations used by the LCN data interfaces (Sections 10 & 11) and the Vintage Routines (Appendix G).
`CM50$LIB:CM50_ERROR_INCLUDE.FOR`	Contains the symbolic names for all CM50S error codes (Appendix A.2).
`CM50$LIB:CM50_FTF_INCLUDE.FOR`	Definitions for all LCN file transfer operations (section 12.4)

9.1.2 Include Files for DDT and ACP Management

These include files are needed by applications that use the CM50S administration calls described in Section 12. DDT and ACP management functions use some shared data structures that are defined in the file CM50_FLAGS_INCLUDE.FOR. Therefore, that file should be included with any program that calls either DDT or ACP functions and must precede the include file(s) defining those specific functions.

`CM50$LIB:CM50_FLAGS_INCLUDE.FOR`	Definitions for the shared data structures in the ACP & DDT Management Interface calls. Must be included prior to either CM50_ACP_INCLUDE.FOR or CM50_DDT_INCLUDE.FOR.
`CM50$LIB:CM50_ACP_INCLUDE.FOR`	Definitions for all ACP Management operations (section 12.1).
`CM50$LIB:CM50_DDT_INCLUDE.FOR`	Definitions for all the DDT Management operations (section 12.2).
`CM50$LIB:CM50_CGDATA_INCLUDE.FOR`	Definitions for all the CG Database retrievals (section 12.3).

9.1.3 Programmatic Interface Flag Parameters

An INTEGER*4 parameter called FLAGS is included in every ACP and DDT management function to control some of the handling options. Some of the flags apply to only the DDT calls, some to only the ACP calls, and some can be used by both. All user-visible flags (as defined in CM50_FLAGS_INCLUDE.FOR) are described below.

- CM50$M_HANDLER—(Bit 0) Indicates that the user has provided a custom exception handler. The default is OFF.
- CM50$M_MSGON—(Bit 1) Prints all diagnostic messages to SYS$OUTPUT. The default is OFF.
- CM50$M_CGRES—(Bit 5) Installs the DDT as CG resident. The default is OFF.
- CM50$M_REBUILD_DDT—(Bit 6) Rebuilds an existing DDT. The default is OFF.
- CM50$M_NO_SOURC_DEBUG—(Bit 7) Produces no error file during DDT build. The default is OFF.
- CM50$M_DMP_DDT_ERRORS—(Bit 8) After building the DDT, sends the error file produced to SYS$OUTPUT. If set, then the CM50$M_NO_SOURCE_DEBUG flag must be OFF.
- CM50$M_ACIDP_ACTIVATE—(Bit 9) Reserved for internal CM50S use.
- CM50$M_WRITE_VT—(Bit 10) Creates the .VT file with write privilege.

All of the flags described above, represent bit masks that can be added together to enable any combination of the flags. These flag values also can be used to see if a particular flag is set. An example is shown below.

```
Flags = CM50$M_HANDLER + CM50$M_MSGON
Return_Status = DDT_SUMMARY(%REF(DDT_Name),
                            %REF(Summary), Flags)
```

9.2 CALLING CONVENTIONS

CM50S interface routines follow the VMS language-independent calling conventions. With the exception of some housekeeping procedures that have no error handling (such as ACPTRP and PRGTRM), they are written as functions.

We recommend that each function call be followed by a logical test of the return_status value. If return_status is true (odd valued), the call was successful (although individual data items may require checking). If not true (even valued status codes), appropriate error handling should be invoked. Note that if the application does not check return_status, the interface routine can be invoked as a called subroutine or procedure in the same manner as VMS system services.

All the calling sequence examples shown in this section are appropriate for FORTRAN programs. However, they do not show the continuation line key (-) required by that language.

Shortword arguments should be declared as INTEGER*2 and must be passed as variables because FORTRAN assumes that any integer constant is INTEGER*4.

Longword arguments should be declared as INTEGER*4.

Character-string and array arguments must be passed with the explicit %REF qualifier (FORTRAN defaults to passing strings by descriptor).

Boolean (True/False) arguments should be declared as INTEGER*2, with a value of 1 for True and 0 for False. (The FORTRAN Logical data type is evaluated as .True. only for a value of -1 which is never returned from the LCN.)

9.3 COMPATIBILITY OF APPLICATION PROGRAM WITH ITS DDTS

Because each application program and its Data Definition Tables (and Multi_Point List structures) are separately built, the system cannot enforce compatibility between a program and any DDT(s) that it uses. That responsibility is up to you.

In particular, it is vital that the dimensions set for data-receiving arrays be large enough to accommodate the maximum data amounts permitted by the named DDT.

Specific points to remember for DDT Get Data and DDT Store Data are

- Dimensions set for each value-type's program array must be equal-to or greater-than the value-type's point count in the referenced DDT. The values can be stored one-for-one or they can be scattered as defined in the DDT. If the program arrays are too small, data or program code may be corrupted (DDT Get Data) or inappropriate data may be exported (DDT Store Data).

- The dimension values for status table arrays must be equal-to or greater-than the total number of points of all types in the referenced DDT because this array is to receive a status code for each returned value, positioned according to its location in the DDT.

9.4 DATA REPRESENTATIONS

Differences between data representations in the VAX and the CG normally are resolved by the CG-VAX Communications Handlers, thus are invisible to the user (Exception: raw data transfers, see heading 10.3). The LCN data formats are:

Real—32 bit floating point matches normal REAL format except that bad values (NaN) from the LCN have the bit pattern for -0. This value will cause a FORTRAN trap if used in an arithmetic statement, so real values returned from the LCN should always be tested (using either the CM50_VALIDN function or the associated value_status_table entry for the value).

Integer—INTEGER*2.

ASCII—CHARACTER*24 variable.

String—CHARACTER*40 variable.

Enumeration—There are two ways to represent LCN enumerations: as CHARACTER*8 ASCII strings (Enumerated) or as INTEGER*2 values (Ordinal). The choice of representation is made when the data transfer is requested, except that **self-defined enumerations should be transferred only as Ordinals**. For information on standard enumerations, see the *Application Module Parameter Reference Dictionary*, *Hiway Gateway Parameter Reference Dictionary*, and *Computer Gateway Parameter Reference Dictionary*. For information on Custom Data Segments, see the *System Control Functions* manual. For information on self-defined enumerations, see Section 2 of the *Hiway Gateway Control Functions* manual.

Time—LCN Internal Time is defined as a record structure /TIME_PT_VALS/ consisting of an INTEGER*4 count of Seconds (since the start of 1979) followed by an INTEGER*2 count of Ticks (tenths of milliseconds). Some of the calls will return LCN External Time, an ASCII string of format MM/DD/YYΔHH:MM:SS, where Δ represents a space. See heading 11.3.3 for time format conversions.

Entity ID—Internally stored as an 64-bit value (array of 4 INTEGER*2) identifying a specific point (Ptid or Internal_id). Also can be retrieved as a CHARACTER*18 variable (External_id) consisting of the up-to-16-character point name followed by the two-character pinid for Network Gateway references.

9.5 COMMONLY MADE ERRORS

- Character string arguments must be the declared length If string constants are used for arguments, they must be padded with spaces (or terminated with a null character). Use of the wrong length for a string probably will result in a FORTRAN runtime error.

- Failure to use the CM50_SET_ACP function (or ACPTRP call) as the first executable program statement of an ACP and/or failure to use the PRGTRM call as the last executable statement of an ACP.

- Attempting to run an application program with unresolved compile or link errors or the use of a DDT that is incomplete or complete with errors.

- Failure to specify array sizes and data types that match DDT definitions.

- Failure to specify all parameters required by the interface routines. Also, failure to specify the correct data type for parameters. Make sure the %REF qualifier is used everywhere it is shown in the examples.

- Attempting to activate an ACP through an ACIDP while the ACP is linked to the VMS DEBUG utility. Use of the DEBUG utility is supported only for execution of ACPs while run interactively from a terminal.

- Terminating an ACP by use of the STOP/IDENTIFIER function of VMS DCL. ACPs should only be aborted through the CM50S Deactivate ACP procedure.

9.6 ERROR DETECTION BY INTERFACE FUNCTIONS

There are three categories of error that can be detected during the execution of a program when using the interface functions. These are indicated through one of these methods:

- Request completion status code

- Individual parameter status codes

- Program abort

The RETURN_STATUS value returned by the Function shows whether or not the request was successfully processed and, if not, what error type was involved. Some typical errors flagged by the return status are

- LCN access problems or data link failure

- ACP installation or mode problems

- Data problems in the call or with a referenced DDT

- Call rules violations

The RETURN_STATUS code follows the standard VAX/VMS condition status code format. In general, even number codes indicate fatal system problems or program bugs, while odd number codes indicate success (code 000000001) or partial success (e.g., code 215000051). See Appendix A.2 for additional information and a listing of all RETURN_STATUS values and their meanings.

Most of the interface calls also return LCN point.parameter values that are to be processed by the calling application program. Accompanying each value (or array) is a status code that must be checked for indications of problems that would invalidate the requested data. See the call arguments STATUS_TABLE or VALUE_STATUS in the individual interface descriptions. There are over 200 different data access-status codes that can be returned. See Appendix A.1 for a listing of these codes.

Some errors in use of the interface routines result in the application program being aborted. An error message is logged at the VAX operator console and is shown on the Universal Station Detail Display for a connected ACIDP. These errors can be of the following types:

- File access errors
- Communication Interface errors
- Format conversion errors
- Various program logic errors

9.7 SUMMARY OF USER-PROGRAM INTERFACES

Heading	Interface Descriptions	Function Names
	Multipoint (DDT) Data Transfers	
10.1.1	DDT Get Data	CM50_DDT_GET CM50_DDT_GETNT
10.1.2	DDT Store Data	CM50_DDT_STORE CM50_DDT_STORENT
10.1.3	Generic DDT Get Data	CM50_DDT_GETGEN
10.1.4	Generic DDT Store Data	CM50_DDT_STOREGEN
10.1.5	Multi-Point List Get Data	CM50_MPL_GET
10.1.6	Multi-Point List Store Data	CM50_MPL_STORE
10.1.7	Generate Multi-Point List	CM50_MPL_GENLIST CM50_MPL_GENTAGS CM50_MPL_GENFILE
10.1.8	Read Multi-Point List	CM50_MPL_READ
10.1.9	Write Multi-Point List	CM50_MPL_WRITE
10.1.10	Create Include File for Multi-Point List	CM50_MPL_GENINCL
	Point List Data Transfers	
10.2.1	Point List Get Values	CM50_GET_PT_LIST
10.2.2	Point List Get by Value Type	
	Real Values	CM50_GET_REALNBR
	Integer Values	CM50_GET_INTNBR
	ASCII Values	CM50_GET_ASC24
	Enumeration Values	CM50_GET_ENUM
	Ordinal Values	CM50_GET_ORD
	Internal IDs	CM50_GET_PTID
	External IDs	CM50_GET_EXID
	Time Values	CM50_GET_TIME
	String Values	CM50_GET_STRI
10.2.3	Point List Store Values	CM50_STORE_PT_LIST
10.2.4	Point List Store by Value Type	
	Real Values	CM50_STORE_REALNBR
	Integer Values	CM50_STORE_INTNBR
	ASCII Values	CM50_STORE_ASC24
	Enumeration Values	CM50_STORE_ENUM
	Ordinal Values	CM50_STORE_ORD
	Internal Ids	CM50_STORE_PTID
	Time Values	CM50_STORE_TIME
	String Values	CM50_STORE_STRI
	Single Point Data Transfers	
10.3.1	Single Point Get Data(External ID)	CM50_GET_ID CM50_GET_TAG
10.3.2	Single Point Store Data(External ID)	CM50_STORE_ID CM50_STORE_TAG
10.3.3	Single Point Get Data (Internal ID)	CM50_GETPT_ID
10.3.4	Single Point Store Data (Internal ID)	CM50_STOREPT_ID
10.3.5	Get LCN Clock Value	CM50_TIMNOW_LCN CM50_TIMNOW_ASC
	Raw Data Transfers	
10.4.1	Raw Data Get	CM50_SPGRAW
10.4.2	Raw Data Store	CM50_SPSRAW
10.4.3	Convert Raw Data	CM50_SPCRAW

Heading	Interface Descriptions	Function Names
	History Data Transfers	
10.5.2	Get History Snapshots (Relative Time)	CM50_DDTHIS_SNAP CM50_DDTHIS_FAST CM50_MPLHIS_SNAP CM50_PTHIS_SNAP
10.5.3	Get History Snapshots (Absolute Time)	CM50_DDTHIS_SNAPT CM50_DDTHIS_FASTT CM50_MPLHIS_SNAPT CM50_PTHIS_SNAPT
10.5.4	Get History Averages (Relative Time)	CM50_DDTHIS_AVER CM50_MPLHIS_AVER CM50_PTHIS_AVER
10.5.5	Get History Averages (Absolute Time)	CM50_DDTHIS_AVERT CM50_MPLHIS_AVERT CM50_PTHIS_AVERT
10.5.6	Get Monthly Averages (Relative Time)	CM50_DDTHIS_MNTH CM50_MPLHIS_MNTH CM50_PTHIS_MNTH
10.5.7	Get Monthly Averages (Absolute Time)	CM50_DDTHIS_MNTHT CM50_MPLHIS_MNTHT CM50_PTHIS_MNTHT
10.5.8	Find History Collection Rate	CM50_DDTHIS_RATE CM50_MPLHIS_RATE CM50_PTHIS_RATE CM50_TAGHIS_RATE
	Text Message Transfers	
10.6.1	Get Message	CM50_GETMSG
10.6.2	Send Message	CM50_STOREMSG
	ACP Execution Support	
11.1.1	ACP Initialization	CM50_SET_ACP
11.1.2	Get ACP Status	GETSTS*
11.1.3	ACP Delay	CM50_ACPDELAY
11.1.4	ACP Hibernate	CM50_HIBER
11.1.5	ACP Termination	PRGTRM*
	Entity Name Conversions	
11.2.1	Convert External to Internal ID	CM50_CONV_PT CM50_CONV_TAG
11.2.2	Convert List of External Ids	CM50_CONV_PT_LIST CM50_CONV_TAG_LIST
	Value Conversions	
11.3.1	Valid Number Check	CM50_VALIDN
11.3.2	Set Bad Value	CM50_SETBAD
11.3.3	Convert Time Values	CM50_TIMLCN_ARY CM50_TIMLCN_ASC CM50_TIMLCN_EURO CM50_TIMLCN_VAXA CM50_TIMLCN_VAXB CM50_TIMARY_LCN CM50_TIMARY_ASC CM50_TIMARY_EURO CM50_TIMARY_VAXA CM50_TIMARY_VAXB CM50_TIMASC_LCN

* GETSTS and PRGTRM do not have a RETURN_STATUS, so they cannot be used as functions, but must be invoked by CALL statements.

Heading	Interface Descriptions	Function Names
11.3.3	Convert Time Values—continued	CM50_TIMASC_ARY CM50_TIMASC_EURO CM50_TIMASC_VAXA CM50_TIMASC_VAXB CM50_TIMEURO_LCN CM50_TIMEURO_ARY CM50_TIMEURO_ASC CM50_TIMEURO_VAXA CM50_TIMEURO_VAXB CM50_TIMVAXA_LCN CM50_TIMVAXA_ARY CM50_TIMVAXA_ASC CM50_TIMVAXA_EURO CM50_TIMVAXA_VAXB CM50_TIMVAXB_LCN CM50_TIMVAXB_ARY CM50_TIMVAXB_ASC CM50_TIMVAXB_EURO CM50_TIMVAXB_VAXA
	ACP Management	
12.1.1	Install an ACP	CM50_ACP_INSTALL
12.1.2	Uninstall an ACP	CM50_ACP_UNINST
12.1.3	Activate (run) an ACP	CM50_ACP_ACT
12.1.4	Deactivate (abort) an ACP	CM50_ACP_DEACTIVATE
12.1.5	Connect an ACP to an ACIDP	CM50_ACP_CONNECT
12.1.6	Disconnect ACP from its ACIDP	CM50_ACP_DISCON
12.1.7	Change ACP installation mode	CM50_ACP_CHG_MODE
12.1.8	Get ACP summary	CM50_ACP_SUM
12.1.9	Get list of ACPs	CM50_ACP_LISTALL
	DDT Management	
12.2.1	Build/Rebuild a DDT	CM50_DDT_BUILD
12.2.2	Delete a DDT	CM50_DDT_DELETE
12.2.3	Get DDT summary information	CM50_DDT_SUM
12.2.4	Get list of DDT summaries	CM50_DDT_LIST
12.2.5	Get DDT detailed information	CM50_DDT_DETAIL
12.2.6	Connect a DDT to an ACIDP	CM50_DDT_CONNECT
12.2.7	Disconnect a DDT from its ACIDP	CM50_DDT_DISCONNECT
12.2.8	Modify DDT prefetch triggers	CM50_DDT_TRIGGERS
12.2.9	Install a DDT as CG resident	CM50_DDT_INSTALL
12.2.10	Remove a DDT from CG residency	CM50_DDT_UNINST
	CG Database Routines	
12.3.1	Get list of resident DDTs	CM50_CG_RDDT
12.3.2	Get list of CRDPs	CM50_CG_CRDP
12.3.3	Get detailed ACIDP information	CM50_CG_ADETAIL
12.3.4	Get list of ACIDPs	CM50_CG_ACIDP
12.3.5	Get LCN Configuration	CM50_CG_CONFIG
	LCN File Transfer Routines	
12.4.1	Read File from LCN	CM50_LCN_READ
12.4.2	Write File to LCN	CM50_LCN_WRITE
12.4.3	List LCN File Attributes	CM50_ATTR_LIST
12.4.4	List LCN Files & Extensions	CM50_FILE_LIST
12.4.5	List LCN Volumes/Directories	CM50_HM_LIST

12.4.6	List LCN Files to Dataout	CM50_FILE_CATALOG
12.4.7	List LCN Volumes to Dataout	CM50_VOLUME_CATALOG
12.4.8	LCN File Copy	CM50_LCN_COPY
12.4.9	LCN File Move	CM50_LCN_MOVE
12.4.10	LCN File Rename	CM50_LCN_RENAME
12.4.11	LCN File Delete	CM50_LCN_DELETE
12.4.12	LCN Directory Maintenance	CM50_LCN_DIRECTORY
12.4.13	LCN Dataout Status	CM50_DATA_OUT
12.4.14	Abort LCN File Transfer	CM50_ABORT_TRANSFER

LCN DATA TRANSFERS (FORTRAN)
Section 10

This section discusses each of the program calls that FORTRAN programs use to transfer data between the host computer and the TDC 3000 Local Control Network.

10.1 MULTIPOINT (DDT) DATA TRANSFERS

The interface routines in this group require the use of separately prepared Data Definition Tables (DDT) that specify which points are to be accessed and what pre/post processing is to be done on data values. See Section 6 for DDT preparation and installation details.

Each DDT may reference a maximum of four different data types. The standard DDT functions assume the data types are grouped into a "normal" order. It is possible to build DDTs with unusual combinations of data types that do not follow these assumptions. These special-case DDTs are tagged as GenIn (Generic Input) or GenOut (Generic Output) and may only be used with the Generic DDT Transfers described in Sections 10.1.3 and 10.1.4. Standard Input and Output DDTs may be used with either the Generic DDT transfers or the traditional DDT data interface routines.

Single elements of parameter arrays (but not whole arrays) can be specified in the DDT.

10.1.1 DDT Get Data Interface

This routine fetches data from the DDT's associated CG or elsewhere on its LCN. The specification of which data is to be fetched and where it is to be stored in the calling program's data arrays is contained in the Data Definition Table referenced by the call.

10.1.1.1 Example FORTRAN Call for DDT Get Data

```
return_status =  CM50_DDT_GET  or CM50_DDT_GETNT
                 (%REF(ddt_name),
                  real_values_array,
                  intg_values_array,
                   or  ptid_values_array,
                   or  time_values_array,
                  %REF(asci_values_array),
                   or  %REF(string_values_array),
                   or  %REF(exid_values_array),
                  %REF(enum_array),
                   or  ord_array,
                  status_table)
```

Use the function name `CM50_DDT_GET` if you want data transformation operations performed by the Table Processor, and `CM50_DDT_GETNT` if you do not want data transformation operations performed (to decrease processing time).

The DDT Get Data call must specify four data types in the order shown (three of these can be dummy arguments that receive no data). Note that there are restrictions on the combinations of data types.

10.1.1.2 Parameter Definitions for DDT Get Data

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a CHARACTER*9 variable that contains the name of the input Data Definition Table to be used.

real_values_array—The name of a Real array where the fetched Real values are to be stored. Bad values are returned as NaN (-0).

intg_values_array—The name of an INTEGER*2 array where the fetched Integer values are to be stored.

ptid_values_array—The name of an array of internal entity ids (where each element is an array of 4 Integer*2 values declared as: INTEGER*2 variable (4,n) where n is the number of ids to be returned.

time_values_array—The name of an array of LCN internal time values (declare as RECORD /TIME_PT_VALS/).

asci_values_array—The name of an array of CHARACTER*24 variables where the fetched ASCII values are to be stored. Bad values are returned as strings of question marks.

string_values_array—The name of an array of CHARACTER*40 variables where the fetched LCN string values are to be stored.

exid_values_array—The name of an array of CHARACTER*18 variables where the fetched external entity names are to be stored.

enum_array—The name of an array of CHARACTER*8 variables where the fetched Enumeration values are to be stored. Bad values are returned as strings of question marks.

ord_array—The name of an INTEGER*2 array where the fetched ordinal values of enumerations are to be stored.

status_table—The name of an INTEGER*2 array for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A.1 for a listing of Data Access error/status codes.

10.1.2 DDT Store Data Interface

This routine sends data to points in the DDT's associated CG or elsewhere on its LCN. The specification of what points are to receive data and the location of data within the calling program's data arrays is contained in the Data Definition Table referenced by the call. Errors encountered during execution of the routine as well as individual point-data errors are returned to the calling program.

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

10.1.2.1 Example FORTRAN Call for DDT Store Data

```
return_status =  CM50_DDT_STORE  or  CM50_DDT_STORENT
                 (%REF(ddt_name),
                  real_values_array,
                  intg_values_array,
                   or  ptid_values_array,
                   or  time_values_array,
                  %REF(asci_values_array),
                   or  %REF(string_values_array),
                  %REF(enum_array),
                   or  ord_array,
                  store_array,
                  status_table)
```

Use the function name CM50_DDT_STORE if you want data transformation operations performed by the Table Processor and CM50_DDT_STORENT if you do not want transformation operations performed (to decrease processing time).

The DDT Store Data call must specify four data types in the order shown (three of these can be dummy arguments that export no data). Note that there are restrictions on the combinations of data types.

10.1.2.2 Parameter Definitions for DDT Store Data

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `status_table` status code for each requested store value must be checked.

ddt_name—The name of a CHARACTER*9 variable that contains the name of the output Data Definition Table to be used in the "Store Data" operation.

real_values_array—The name of a Real array that contains the Real values to be stored.

intg_values_array—The name of an INTEGER*2 array that contains the Integer values to be stored.

ptid_values_array—The name of an array of internal entity ids (where each element is an array of 4 Integer*2 values declared as: INTEGER*2 variable (4,n) where n is the number of ids to be stored.

time_values_array—The name of an array of LCN internal time values (declare as RECORD /TIME_PT_VALS/).

asci_values_array—The name of an array of CHARACTER*24 variables that contains the ASCII values to be stored.

string_values_array—The name of an array of CHARACTER*40 variables where LCN string values are stored.

enum_array—The name of an array of CHARACTER*8 variables that contains the Enumeration values to be stored. Use of enumeration strings by Store Data is limited to standard enumerations (including Custom Data Segments). All self-defined enumerations (such as digitals) must be accessed through their ordinal values.

ord_array—The name of an INTEGER*2 array that contains the Ordinal values of Enumerations to be stored.

store_array—The name of an INTEGER*2 array that contains a control code entry for each value to be stored. These codes control what—if any—value is to be stored. The store code values are

0 – Store the value from the Values Array
1 – Store the bad value representation instead
2 – Do not store any value.

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

status_table—The name of an INTEGER*2 array for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A for a listing of Data Access error/status codes.

10.1.3 Generic DDT Get Data Interface

This routine fetches data for any Input or Generic Input DDT. The specification of which data is to be fetched and where it is to be stored in the calling program's data arrays is contained in the Data Definition Table referenced by the call.

10.1.3.1 Example FORTRAN Call for Generic DDT Get

```
return_status =  CM50_DDT_GETGEN
                 (%REF(ddt_name),
                  %REF(values_array1),
                  %REF(values_array2),
                  %REF(values_array3),
                  %REF(values_array4),
                  status_table,
                  tbl_proc)
```

10.1.3.2 Parameter Definitions for Generic DDT Get

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a CHARACTER*9 variable that contains the name of the Data Definition Table to be used.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array where the fetched values are to be stored. The data type for each array must match the corresponding data type in the DDT definition. Each array may be declared to match the specific type and number of elements returned by the DDT; if the DDT contains fewer than 4 data types, the unused arguments may be omitted (but the correct number of commas is required).

status_table—The name of an INTEGER*2 array for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A.1 for a listing of Data Access error/status codes.

tbl_proc—The name of an INTEGER*2 that determines whether or not table processing is to be suppressed. If tbl_proc is set to 1, all table processing (saving values to disk and/or data transformations) will be suppressed. Use a value of 0 for normal processing.

10.1.4 Generic DDT Store Data Interface

This routine sends data to points defined in any Output or Generic Output DDT. The specification of what points are to receive data and the location of data within the calling program's data arrays is contained in the Data Definition Table referenced by the call. Errors encountered during execution of the routine as well as individual point-data errors are returned to the calling program.

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* Manual for other write access restrictions.

10.1.4.1 Example FORTRAN Call for Generic DDT Store

```
return_status =  CM50_DDT_STOREGEN
                 (%REF(ddt_name),
                  %REF(values_array1),
                  %REF(values_array2),
                  %REF(values_array3),
                  %REF(values_array4),
                  store_array,
                  status_table,
                  tbl_proc)
```

10.1.4.2 Parameter Definitions for Generic DDT Store

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `status_table` status code for each requested store value must be checked.

ddt_name—The name of a CHARACTER*9 variable that contains the name of the Data Definition Table to be used in the "Store Data" operation.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array that contains the data to be stored. The data type for each array must match the corresponding data type in the DDT definition. Each array may be declared to match the specific type and number of elements returned by the DDT; if the DDT contains fewer than 4 data types, the unused arguments may be omitted (but the correct number of commas is required).

store_array—The name of an INTEGER*2 array that contains a control code entry for each value to be stored. These codes control what—if any—value is to be stored. The store code values are

0 – Store the value from the Values Array
1 – Store the bad value representation instead
2 – Do not store any value.

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

status_table—The name of an INTEGER*2 array for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A for a listing of Data Access error/status codes.

tbl_proc—The name of an INTEGER*2 that determines whether or not table processing is to be suppressed. If tbl_proc is set to 1, all table processing (saving values to disk and/or data transformations) will be suppressed. Use a value of 0 for normal processing.

10.1.5 Multi-Point List Get Data Interface

This routine fetches data for the LCN tags specified in an internal data block. An internal Data Block is a memory-resident equivalent of a DDT. The specification of which data is to be fetched and where it is to be stored in the calling program's data arrays can be prepared using any of the generate MPL routines (see 10.1.7) or you can read in a DDT from its disk file (see 10.1.8).

10.1.5.1 Example FORTRAN Call for Multi-Point List Get

```
return_status =  CM50_MPL_GET
                 (%REF(mpl_name),
                  %REF(acidp_name),
                  %REF(values_array1),
                  %REF(values_array2),
                  %REF(values_array3),
                  %REF(values_array4),
                  status_table,
                  cg_port_num)
```

10.1.5.2 Parameter Definitions for Multi-Point List Get

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (CM50_ACP_RUN)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (CM50_LCN_PART)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

mpl_name—The name of a Multi-Point List structure defining the data to be retrieved. This should be declared as RECORD /CM50_IDB_REC/.

acidp_name—A CHARACTER*16 variable containing the name of an ACIDP. If the ACIDP is spaces, then the data will be retrieved without any ACIDP controls. If an ACIDP is named, then the data access will be completed only if that ACIDP is in RUN state.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array where the fetched values are to be stored. The data type for each array must match the corresponding data type in the MPL definition. Each array may be declared to match the specific type and number of elements returned by the MPL; if the MPL contains fewer than 4 data types, the unused arguments may be omitted (but the correct number of commas is required).

status_table—The name of an INTEGER*2 array for the storage of returned point-related error/status information. A value_status code is returned for each requested tag in the list. See Appendix A.1 for a listing of Data Access error/status codes.

cg_port_num—The name of an INTEGER*2 (with a value of 1-4) identifying the CG to be accessed.

10.1.6 Multi-Point List Store Data Interface

This routine stores data for the LCN tags specified in an internal data block. An internal Data Block is a memory-resident equivalent of a DDT. The specification of which tags are to receive data and the location of the values within the calling program's data arrays can be prepared using any of the generate MPL routines (see 10.1.7) or you can read in a DDT from its disk file (see 10.1.8).

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

10.1.6.1 Example FORTRAN Call for Multi-Point List Store

```
return_status =  CM50_MPL_STORE
                 (%REF(mpl_name),
                  %REF(acidp_name),
                  %REF(values_array1),
                  %REF(values_array2),
                  %REF(values_array3),
                  %REF(values_array4),
                  store_array,
                  status_table,
                  cg_port_num)
```

10.1.6.2 Parameter Definitions for Multi-Point List Store

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (CM50_ACP_RUN)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (CM50_LCN_PART)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

mpl_name—The name of a Multi-Point List structure defining the data to be stored. This should be declared as RECORD /CM50_IDB_REC/.

acidp_name—A CHARACTER*16 variable containing the name of an ACIDP. If the ACIDP is spaces, then the ACIDP currently connected to the ACP will control the data transfer. If an ACIDP is named, then the data access will be completed only if that ACIDP is in RUN state.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array that contains the data to be stored. The data type for each array must match the corresponding data type in the MPL definition. Each array may be declared to match the specific type and number of elements returned by the MPL; if the MPL contains fewer than 4 data types, the unused arguments may be omitted (but the correct number of commas is required).

store_array—The name of an INTEGER*2 array that contains a control code entry for each value to be stored. These codes control what—if any—value is to be stored. The store code values are

0 – Store the value from the Values Array
1 – Store the bad value representation instead
2 – Do not store any value
16386 - Store IEEE negative infinity instead of Real value
16387 - Store IEEE positive infinity instead of Real value

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

status_table—The name of an INTEGER*2 array for the storage of returned point-related error/status information. A value_status code is returned for each requested tag in the list. See Appendix A.1 for a listing of Data Access error/status codes.

cg_port_num—The name of an INTEGER*2 (with a value of 1-4) identifying the CG to be accessed.

10.1.7 Generate Multi-Point List

These routines generate an Internal data block for transfer arrays of up to four data types between the LCN and host computer. Internal data blocks are subject to exactly the same restrictions as DDTs (see Table 6-1).

A Multi-Point List may be generated from either a set of ID Block Arrays (such as those produced using the Convert Lists calls—see section 11.2.2), or a text file of type declarations and tag names, or an array of text entries.

NOTE

The arrays of internal point.parameter addresses need to be rebuilt and the program(s) using them need to be recompiled whenever the LCN database is changed in a significant manner, such as by the rebuild or deletion of data points referenced in the address array.

10.1.7.1 Example FORTRAN Calls to Generate Multi-Point Lists

To combine point lists, use:

```
return_status =     CM50_MPL_GENLIST
                    (list_size,
                     id_block_arr1,
                     id_block_arr2,
                     id_block_arr3,
                     id_block_arr4,
                     mpl_name)
```

When the external ids are expressed as a Tag name list, use:

```
return_status =     CM50_MPL_GENTAGS
                    (%ref(tagname_arr),
                     number_of_values,
                     mpl_name,
                     cg_port_num
                     return_arr)
```

When the external ids are contained in a Text file, use:

```
return_status =     CM50_MPL_GENFILE
                    (%ref(tag_file),
                     mpl_name,
                     cg_port_num
                     return_arr)
```

10.1.7.2 Parameter Definitions for Generate Multi-Point Lists

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `return_array` status code for each returned value must be checked.

tagname_arr—The name of an array of up to 304 CHARACTER*40 variables that contains the ASCII tagname (formatted as Point.Param, optionally with the parameter index enclosed in parentheses) of the LCN entity for which the internal ID is to be obtained. All tagnames of the same data type must be grouped together and different data types must be separated by the reserved "tag" of:

`**NEWΔTYPE=type` where Δ is a required space,

and "type" (starting in position 12) is one of the following:

REAL	real number
INTE	integer
ASCI	24 character ASCII
ENUM	enumeration
ORDN	ordinal
PTID	internal entity id
EXID	external entity id
TIME	lcn time type
STRI	40 CHARACTER* variable

If the first item in the array does not contain "**NEW TYPE=" in positions 1 through 11, then the first set of tagnames is assumed to identify Real numbers.

number_of_values —The name of an INTEGER*2 specifying the number of tags defined in the tagname_arr. The maximum number of values is 304.

tag_file—The CHARACTER*80 pathname of a text file whose content is a tagname_array, with each line containing either a valid tagname or a "**NEW TYPE=" tag as described above. Note that if the file is created by a FORTRAN program, the OPEN statement that creates the file should specify:

```
FORM = UNFORMATTED, CARRIAGECONTROL = NONE
```

list_size —The name of an array of 4 INTEGER*2 values that specify the number of tags defined in each id_block_arr. The maximum number of values is 300.

id_block_arr*n* —(where **n** is 1 to 4) The name of an array of point addresses in internal format (declare as array of `RECORD /ID_BLOCK_STRUCT/`) from which values will be requested. If fewer than four ID Block Arrays are required, the unused arguments may be omitted (but the correct number of commas is required).

mpl_name—The name of a Multi-Point List structure where the generated definition is to be stored. This should be declared as RECORD /CM50_IDB_REC/.

cg_port_num—An INTEGER*2 (with a value of 1-4) identifying the CG to be accessed.

return_arr—The name of an array of up to 304 INTEGER*4 to receive the status of the conversion of each tag and data type declaration, including field type records. See Appendix A.2 for an explanation and a listing of all assigned return code values.

10.1.8 Read Multi-Point List

This routine reads an MPL from a disk file that has been created using either the DDT Build procedures or the Write Multi-Point List routine.

10.1.8.1 Example FORTRAN Call to Read Multi-Point Lists

```
return_status =      CM50_MPL_READ
                     (%REF(idb_file),
                      mpl_name)
```

10.1.8.2 Parameter Definitions for Read Multi-Point List

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

idb_file—The CHARACTER*80 pathname of a file containing the Multi-Point List. To reference a DDT, use the pathname of `CM50$DDT:ddtname.II`. If no extension is specified, the default of .MPL will be used.

mpl_name—The name of a Multi-Point List structure in memory. This should be declared as RECORD /CM50_IDB_REC/.

10.1.9 Write Multi-Point List

This routine creates a disk file containing an MPL produced through the Generate Multi-Point List interface (section 10.1.7).

10.1.9.1 Example FORTRAN Call to Write Multi-Point Lists

```
return_status =      CM50_MPL_WRITE
                     (%REF(idb_file),
                      mpl_name)
```

10.1.9.2 Parameter Definitions for Write Multi-Point List

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

idb_file—The CHARACTER*80 pathname of a file to contain the Multi-Point List. If a file by that name already exists, a new version will be created. By default, an extension of .MPL will be used. The use of .II as an extension is prohibited because that extension is reserved for DDTs. It is the user's responsibility to purge obsolete versions.

mpl_name—The name of an Multi-Point List structure in memory. This should be declared as RECORD /CM50_IDB_REC/.

10.1.10 Create Include File for Multi-Point List

This routine creates a disk file containing the text description of an MPL in a format suitable for use as an include file for a FORTRAN source program. The MPL should be previously produced through the Generate Multi-Point List interface (see heading 10.1.7).

10.1.10.1 Example FORTRAN Call to Generate a Multi-Point List Include File

```
return_status =     CM50_MPL_GENINCL
                    (%REF(mpl_name),
                     %REF(text_file),
                     %REF(Language))
```

10.1.10.2 Parameter Definitions for Generate Multi-Point List Include File

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

mpl_name—The name of a Multi-Point List structure in memory. This should be declared as RECORD /CM50_IDB_REC/.

text_file—The CHARACTER*80 pathname of the include file to be written. If a file by that name already exists, a new version will be created. No default extension is provided. It is the users responsibility to purge obsolete versions.

language— A CHARACTER*1 code identifying the format of the include file:

'P' = Pascal
'C' = C
'F' = FORTRAN

Any other value will default to FORTRAN.

10.2 POINT LIST TRANSFERS

These routines enable you to address multiple points with a single call without needing to build DDT tables. In the place of a DDT reference, you will have to provide a pointer to an array of "internal" point.parameter addresses. These internal addresses can be obtained by conversion calls at program runtime (see heading 11.2), or in advance by creating an include file through the Utility MAKEINC (see heading 7.2).

10.2.1 Point List Get Values Interface

This function returns data values to up to 300 points on the LCN without using DDT tables. The specification of which data is to be fetched and where it is to be stored is contained in the call.

Use of Internal Point-parameter IDs is required. Individual elements of parameter arrays can be specified by repeating the point.parameter address using a changed parameter index. The data type of the values is determined from the Internal Id of the first point in the list.

10.2.1.1 Example FORTRAN Call for Point List Get Values

```
return_status =  CM50_GET_PT_LIST
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(values_array),
                  status_table,
                  number_of_values)
```

10.2.1.2 Parameter Definitions for Point List Get Values

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (CM50_ACP_RUN)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (CM50_LCN_PART)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

cg_port_num—The name of an INTEGER*2 identifying the CG (1-4) to be accessed.

priority—The name of an INTEGER*2 that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

acidp_name—The name of a CHARACTER*16 variable that contains the name of an ACIDP. If the ACIDP name value is blank (all spaces), then the data is retrieved without any ACIDP controls. If an ACIDP is named, then the data access is completed only if that ACIDP is in RUN state.

point_list_array—The name of an array of point addresses in internal format (declare as an array of `RECORD/ID_BLOCK_STRUCT/`) from which the values are requested. See the function CM50_CONV_PT_LIST or CM50_CONV_TAG_LIST (heading 11.2.2) for additional information.

values_array—The name of an array from which the individual values are to be returned. This array is passed by reference, so no type checking is done. It is the application's responsibility to insure that this argument is declared in a manner that is compatible with the value type of the list.

status_table—The name of an INTEGER*2 array where the value status for individual point values are to be stored. See Appendix A.1 for a listing of Data Access error/status codes.

number_of_values—The name of an INTEGER*2 that specifies the actual number of values (300 or less) to be processed.

10.2.2 Point List Get by Value Type

These functions are identical to the CM50_GET_PT_LIST function, except that the value type is part of the function name and the generic "values_array" argument is replaced by an array whose data type explicitly matches the specified data type.

These specific functions and their corresponding value arrays are described below. Refer to heading 10.2.1.2 for explanations of all of the other arguments.

10.2.2.1 FORTRAN Call for Point List Get Real Values

```
return_status =  CM50_GET_REALNBR
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  real_values_array,
                  status_table,
                  number_of_values)
```

real_values_array—The name of a Real array where the individual point values are to be stored. Bad values are left as NaN (-0).

10.2.2.2 FORTRAN Call for Point List Get Integer Values

```
return_status =  CM50_GET_INTNBR
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  intg_values_array,
                  status_table,
                  number_of_values)
```

intg_values_array—The name of an INTEGER*2 array where the individual point values are to be stored.

10.2.2.3 FORTRAN Call for Point List Get ASCII Values

```
return_status =  CM50_GET_ASC24
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(asci_values_array),
                  status_table,
                  number_of_values)
```

asci_values_array—The name of an array of CHARACTER*24 variables where the individual point values are to be stored.

10.2.2.4 FORTRAN Call for Point List Get Enumerated Values

```
return_status =  CM50_GET_ENUM
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(aenm_values_array),
                  status_table,
                  number_of_values)
```

aenm_values_array—The name of an array of CHARACTER*8 variables where the individual point values are to be stored.

10.2.2.5 FORTRAN Call for Point List Get Ordinal Values

```
return_status =  CM50_GET_ORD
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  oenm_values_array,
                  status_table,
                  number_of_values)
```

oenm_values_array—The name of an INTEGER*2 array where the ordinal values of the fetched enumerations are to be stored.

10.2.2.6 FORTRAN Call for Point List Get Internal IDs

```
return_status =  CM50_GET_PTID
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  ptid_values_array,
                  status_table,
                  number_of_values)
```

ptid_values_array—The name of an array of 4-word internal entity IDs where the individual point values are to be stored.

10.2.2.7 FORTRAN Call for Point List Get External IDs Values

```
return_status =  CM50_GET_EXID
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(exid_values_array),
                  status_table,
                  number_of_values)
```

exid_values_array—The name of an array of CHARACTER*18 variables to hold the returned values.

10.2.2.8 FORTRAN Call for Point List Get Time Values

```
return_status =  CM50_GET_TIME
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  time_values_array,
                  status_table,
                  number_of_values)
```

time_values_array—The name of an array of TIME_PT_VALS records (Integer*4 Seconds followed by Integer*2 Ticks) where the individual point values are to be stored.

10.2.2.9 FORTRAN Call for Point List Get String Values

```
return_status =  CM50_GET_STRI
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(stri_values_array),
                  status_table,
                  number_of_values)
```

stri_values_array—The name of an array of CHARACTER*40 variables where the individual point values are to be stored.

10.2.3 Point List Store Values Interface

This function exports data values to up-to-300 points on the LCN without using DDT tables. The specification of which data is to be fetched and where it is to be stored is contained in the call.

Use of Internal Point-parameter IDs is required. Individual elements of parameter arrays can be specified by repeating the point.parameter address using a changed parameter index. The data type of the values is determined from the Internal Id of the first point in the list. Note: Entity ids can only be stored using their internal form.

10.2.3.1 Example FORTRAN Call for Point List Store Values

```
return_status =  CM50_STORE_PT_LIST
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(values_array),
                  store_code_table,
                  status_table,
                  number_of_values)
```

10.2.3.2 Parameter Definitions for Point List Store

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (CM50_ACP_RUN)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (CM50_LCN_PART)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

cg_port_num—The name of an INTEGER*2 identifying the CG (1-4) to be accessed.

priority—The name of an INTEGER*2 that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

acidp_name—The name of a CHARACTER*16 variable that contains the name of an ACIDP. If the ACIDP name value is blank (all spaces), then the ACIDP currently connected to the ACP will control the data transfer. The data access is completed only if the named or implied ACIDP is in RUN state.

point_list_array—The name of an array of point addresses in internal format (declare as an array of `RECORD/ID_BLOCK_STRUCT/`) from which the values are requested. See the function CM50_CONV_PT_LIST or CM50_CONV_TAG_LIST (heading 11.2.2) for additional information.

values_array—The name of an array from which the individual values are to be obtained. This array is passed by reference, so no type checking is done. It is the applications responsibility to insure that this argument is declared in a manner that is compatible with the value type of the list.

store_code_table—The name of an INTEGER*2 array where the calling program has stored a control code for each value to be stored. These codes control what—if any—value is to be stored. The store code values are:

0 = Store the value from Values Array
1 = Store the bad value representation instead of Real or ASCII value
2 = Do not store any value
16386 = Store IEEE negative infinity instead of Real value
16387 = Store IEEE positive infinity instead of Real value

status_table—The name of an INTEGER*2 array where the value status for each individual point value is to be stored. See Appendix A.1 for interpretation of values.

number_of_values—The name of an INTEGER*2 that specifies the actual number of values (300 or less) to be processed.

10.2.4 Point List Store by Value Type

These functions are identical to the CM50_STORE_PT_LIST function, except that the value type is part of the function name and the generic "values_array" argument is replaced by an an array whose data type explicitly matches the specified data type.

These specific functions and their corresponding value arrays are described below. Refer to heading 10.2.3.2 for explanations of all of the other arguments.

10.2.4.1 FORTRAN Call for Point List Store Real Values

```
return_status =  CM50_STORE_REALNBR
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  real_values_array,
                  store_code_table,
                  status_table,
                  number_of_values)
```

real_values_array—The name of a Real array from which the individual values are to be obtained.

10.2.4.2 FORTRAN Call for Point List Store Integer Values

```
return_status =  CM50_STORE_INTNBR
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  intg_values_array,
                  store_code_table,
                  status_table,
                  number_of_values)
```

intg_values_array—The name of an INTEGER*2 array from which the individual values are to be obtained.

10.2.4.3 FORTRAN Call for Point List Store ASCII Values

```
return_status =  CM50_STORE_ASC24
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(asci_values_array),
                  store_code_table,
                  status_table,
                  number_of_values)
```

asci_values_array—The name of an array of CHARACTER*24 variables from which the individual point values are to be obtained.

10.2.4.4 FORTRAN Call for Point List Store Enumerated Values

```
return_status =  CM50_STORE_ENUM
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(aenm_values_array),
                  store_code_table,
                  status_table,
                  number_of_values)
```

aenm_values_array—The name of an array of CHARACTER*8 variables from which the individual point values are to be obtained.

10.2.4.5 FORTRAN Call for Point List Store Ordinal Values

```
return_status =  CM50_STORE_ORD
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  oenm_values_array,
                  store_code_table,
                  status_table,
                  number_of_values)
```

oenm_values_array—The name of an INTEGER*2 array from which the individual point values are to be obtained.

10.2.4.6 FORTRAN Call for Point List Store Internal IDs

```
return_status =  CM50_STORE_PTID
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  ptid_values_array,
                  store_code_table,
                  status_table,
                  number_of_values)
```

ptid_values_array—The name of an array of 64-bit (array of 4 INTEGER*2 values) Internal entity IDs from which the individual point values are to be obtained.

10.2.4.7 FORTRAN Call for Point List Store Time Values

```
return_status =  CM50_STORE_TIME
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(time_values_array),
                  store_code_table,
                  status_table,
                  number_of_values)
```

time_values_array—The name of an array of TIME_PT_VALS records (Integer*4 Seconds followed by Integer*2 Ticks) from which the individual point values are to be obtained.

10.2.4.8 FORTRAN Call for Point List Store String Values

```
return_status =  CM50_STORE_STRI
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(stri_values_array),
                  store_code_table,
                  status_table,
                  number_of_values)
```

stri_values_array—The name of an array of CHARACTER*40 variables from which the individual point values are to be obtained.

10.3 SINGLE POINT DATA TRANSFERS

The interface routines in this group Get or Store values from or to one named point.parameter (or parameter array) at a time. For parameter arrays, up to the whole array is accessed. The External ID version of Get Single Point is also used to get LCN date and time.

10.3.1 Single Point Get Data (External ID) Interface

This routine fetches data for a single point from a specified CG or elsewhere on its LCN. The specification of which data is to be fetched and where it is to be stored is contained in the call. For parameter arrays, either a single element, the whole array, or an array subset starting with the first element can be specified. The point may be identified by either a combination of point and parameter names or by a single tag name.

10.3.1.1 Example FORTRAN Calls for Single Point Get

Using point and parameter names as separate variables:

```
return_status =  CM50_GET_ID
                 (%REF(entity),
                  %REF(param),
                  param_ix,
                  %REF(val_loc),
                  val_st,
                  val_typ,
                  cg_port_num)
```

Using a complete tag name:

```
return_status =  CM50_GET_TAG
                 (%REF(tag_name),
                  %REF(val_loc),
                  val_st,
                  val_typ,
                  cg_port_num)
```

10.3.1.2 Parameter Definitions for Single Point Get

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000051 (CM50_LCN_PART)—the `val_st` status code for each returned value must be checked.

215000322 (CM50_ACC_SIZE)—the array size specified by `param_ix` is larger than the actual size.

tag_name—The name of a CHARACTER*40 variable that identifies the LCN value(s) to be retrieved. The tag name is formatted as "point.param (param_ix)".

entity—The name of a CHARACTER*20 variable that contains the ASCII Point ID. It should contain a point name of up to 16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter for Network Gateway routing.

param—The name of a CHARACTER*8 variable that contains the ASCII name of a parameter (or parameter array) from which the value(s) is retrieved.

param_ix—The name of an INTEGER*2 positive value. Use of this value is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, 5, 13, 15, 17 or 19, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of `param_ix` is used as an index and must be greater than zero. If the parameter being accessed is not an array type, `param_ix` must be zero.

When `val_typ` is 7, 8, 9, 10, 14, 16, 18 or 20, a whole array (or a subset of the array starting with the first element) is to be accessed and `param_ix` is used to specify the number of elements to be accessed—If `param_ix` is smaller than the actual array size, only that number of elements is returned; if it is larger than the actual array size, no elements are returned and the return_status value is CM50_ACC_SIZE.

val_loc—The name of a program variable where the value(s) are to be stored. The type of variable must match what is declared in `val_typ`.

val_typ	val_loc type
1	Real (REAL)
2	Integer (INTEGER*2)
3	ASCII (CHARACTER*24)
4	Enumeration (CHARACTER*8)
5	Ordinal (INTEGER*2)
6	External Time (CHARACTER*18), see heading 10.3.5
7	Array [1..n] of Real
8	Array [1..n] of Integer
9	Array [1..n] of Enumeration
10	Array [1..n] of Ordinal
13	Internal_id (ARRAY [1..4] OF INTEGER*2)
14	Array [1..n] of Internal_id
15	External_id (CHARACTER*18)
16	Array [1..n] External_id
17	Internal Time (TIME_PT_VALS record)
18	Array [1..n] of Internal Time
19	String (CHARACTER*40)
20	Array [1..n] of String

val_st—The name of an INTEGER*2 where point-related status information is to be stored. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (215000051). See Appendix A.1 for a listing of Data-Access error/status codes. When `val_typ` specifies an array, `val_st` refers to status of the whole array.

val_typ—The name of an INTEGER*2 that contains a number that designates value type of the accessed parameters as listed for **val_loc.**

cg_port_num—The name of an INTEGER*2 (1-4) identifying the CG to be accessed.

10.3.2 Single Point Store Data (External ID) Interface

This routine stores data to a single point in a specified CG or elsewhere on its LCN. The specification of where the data is to be found and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

To use this call the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* Manual for other write access restrictions.

10.3.2.1 Example FORTRAN Calls for Single Point Store

Using point and parameter names as separate variables:

```
return_status =  CM50_STORE_ID
                 (%REF(entity),
                  %REF(param),
                  param_ix,
                  %REF(val_loc),
                  val_typ,
                  store_cd,
                  store_st)
```

Using a complete tag name:

```
return_status =  CM50_STORE_TAG
                 (%REF(tag_name),
                  %REF(val_loc),
                  val_typ,
                  store_cd,
                  store_st)
```

10.3.2.2 Parameter Definitions for Single Point Store

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `store_st` status code for each returned value must be checked.

tag_name—The name of a CHARACTER*40 variable that identifies the LCN value(s) to be stored. The tag name is formatted as "point.param (param_ix)".

entity—The name of a CHARACTER*20 variable that contains the ASCII Point ID. It should contain a point name of up to 16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter for Network Gateway routing.

param—The name of CHARACTER*8 variable that contains the ASCII parameter name for the point.parameter where the value is to be stored.

param_ix—The name of an INTEGER*2 positive value. Use of this value is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, 5, 13, 17, or 19, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of `param_ix` is used as an index and must be greater than zero. If the parameter being accessed is not an array type, `param_ix` must be zero.

When `val_typ` is 7, 8, 9, 10, 14, 18, or 20, a whole array is to be accessed and `param_ix` is used to specify the number of array elements—If `param_ix` does not match the actual array size, no elements are stored and `return_status` value is 5 with a `store_st` indicating an invalid array size.

val_loc—The name of a program variable containing the value(s) to be stored. The type of variable must match what is declared in `val_typ`.

val_typ	val_loc type
1	Real (REAL)
2	Integer (INTEGER*2)
3	ASCII (CHARACTER*24)
4	Enumeration (CHARACTER*8)
5	Ordinal (INTEGER*2)
6	External Time (CHARACTER*18), see heading 10.3.5
7	Array [1..n] of Real
8	Array [1..n] of Integer
9	Array [1..n] of Enumeration
10	Array [1..n] of Ordinal
13	Internal_id (ARRAY [1..4] OF INTEGER*2)
14	Array [1..n] of Internal_id
17	Internal Time (TIME_PT_VALS record)
18	Array [1..n] of Internal Time
19	String (CHARACTER*40)
20	Array [1..n] of String

val_typ—The name of an INTEGER*2 that contains a number that designates value type of the accessed parameters as listed for **val_loc.**

store_cd—Name of an INTEGER*2 that contains a code that allows the substitution of a bad value representation in place of the provided value(s). The store code values are

0 = Store the data value(s) provided
1 = Store the bad value representation instead

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

store_st—The name of an INTEGER*2 to contain point-related store status information on completion. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (215000051). See Appendix A.1 for a listing of Data-Access error/status codes. When the `val_typ` is an array, `store_st` refers to status of the whole array.

10.3.3 Single Point Get Data (Internal ID) Interface

This routine fetches data for a single point from the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the CM50_CONV_PT or CM50_CONV_TAG interface, see 11.2.1) reduces the overhead required for repetitive single-point requests.

The specification of which data is to be fetched and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

10.3.3.1 Example FORTRAN Call for Single Point Get

```
return_status =  CM50_GETPT_ID
                 (%REF(id_block),
                  %REF(val_loc),
                  val_st,
                  cg_port_num)
```

10.3.3.2 Parameter Definitions for Single Point Get

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `val_st` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (declare as `RECORD/ID_BLOCK_STRUCT/`) that contains the internal ID data block obtained by a previous Convert External to Internal ID call. When the data is of array type, that call returns the array size in word 7 of the ID block. Thus, if you wish to get less than the entire array you can change the parameter qualifier in the seventh word of the ID block to be smaller than the actual array size. Do not change any other words in the ID block. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

val_loc—The name of a program variable where the value is to be stored. The type of variable must match what was declared in `val_typ` in the earlier Convert ID call.

val_typ	val_loc type
1	Real (REAL)
2	Integer (INTEGER*2)
3	ASCII (CHARACTER*24)
4	Enumeration (CHARACTER*8)
5	Ordinal (INTEGER*2)
6	External Time (CHARACTER*18), see heading 10.3.5
7	Array [1..n] of Real
8	Array [1..n] of Integer
9	Array [1..n] of Enumeration
10	Array [1..n] of Ordinal
13	Internal_id (ARRAY [1..4] OF INTEGER*2)
14	Array [1..n] of Internal_id
15	External_id (CHARACTER*18)
16	Array [1..n] External_id
17	Internal Time (TIME_PT_VALS record)
18	Array [1..n] of Internal Time
19	String (CHARACTER*40)
20	Array [1..n] of String

val_st—The name of an INTEGER*2 where point-related status information is to be stored. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (CM50_LCN_PART). When the `val_typ` specifies an array, `val_st` refers to status of the whole array.

cg_port_num—The name of an INTEGER*2 identifying the CG (1-4) to be accessed.

10.3.4 Single Point Store Data (Internal ID) Interface

This routine stores data to a single point in the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the CM50_CONV_PT or CM50_CONV_TAG interface, see 11.2.1) reduces the overhead required for repetitive single-point requests.

The specification of where the data is found and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

To use this function the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

10.3.4.1 Example FORTRAN Call for Single Point Store

```
return_status =  CM50_STOREPT_ID
                 (%REF(id_block),
                  %REF(val_loc),
                  store_cd,
                  store_st)
```

10.3.4.2 Parameter Definitions for Single Point Store

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `store_st` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (for FORTRAN declare as `RECORD/ID_BLOCK_STRUCT/`) that contains the internal ID data block obtained by a previous Convert External to Internal ID call. Do not change any words in the ID block. If the array size is changed, the array is not stored and the `return_status` value is 215000051, with a `store_st` that indicates an invalid array size. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

val_loc—The name of a program variable that contains the value to be stored. The type of variable must match what was declared in `val_typ` in the earlier Convert ID call.

val_typ	val_loc type
1	Real (REAL)
2	Integer (INTEGER*2)
3	ASCII (CHARACTER*24)
4	Enumeration (CHARACTER*8)
5	Ordinal (INTEGER*2)
6	External Time (CHARACTER*18), see heading 10.3.5
7	Array [1..n] of Real
8	Array [1..n] of Integer
9	Array [1..n] of Enumeration
10	Array [1..n] of Ordinal
13	Internal_id (ARRAY [1..4] OF INTEGER*2)
14	Array [1..n] of Internal_id
17	Internal Time (TIME_PT_VALS record)
18	Array [1..n] of Internal Time
19	String (CHARACTER*40)
20	Array [1..n] of String

store_cd—The name of an INTEGER*2 that contains a code that allows the substitution of a bad value representation in place of the provided value(s). The store code values are

0 = Store the data value(s) provided
1 = Store the bad value representation instead

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

store_st—The name of an INTEGER*2 where point-related status information is to be stored. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (CM50_LCN_PART). When the `val_typ` specifies an array, `store_st` refers to status of the whole array.

10.3.5 Get LCN Clock Value Interface

The current date and time as kept by the LCN, can be obtained in either internal or ASCII format. The internal format is a 4-byte integer count of the number of seconds since January 1, 1979. The ASCII format is MM/DD/YYΔHH:MM:SSΔ (where Δ is used to indicate a space).

10.3.5.1 Example FORTRAN Calls to Get the LCN Clock

Internal Time Format:

```
       return_status =    CM50_TIMNOW_LCN
                          (Integer_Clock,
                           cg_port_num)
```

ASCII Time Format:

```
       return_status =    CM50_TIMNOW_ASC
                          (%REF(ASCII_Clock),
                           cg_port_num)
```

10.3.5.2 Parameter Definitions for Get LCN Clock

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

Integer_clock—The name of an INTEGER*4 where the clock value, in seconds, is to be stored.

ASCII_clock—The name of a CHARACTER*18 variable where the clock value, formatted as 'MM/DD/YY hh:mm:ss ', is to be stored.

cg_port_num—The name of an INTEGER*2 (with a value of 1-4) identifying the CG to be accessed.

10.4 RAW DATA TRANSFERS

The interface routines in this group get, store, and convert **only** LCN **Real data arrays** in LCN format. Each request works only with a single data point's parameter array. These functions allow you to pass Real data arrays from one LCN to another without needing to go through the LCN/VAX data conversions.

10.4.1 Get Raw Data Interface

This function fetches data for a single point from the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the Convert External to Internal ID interface, see 11.2.1) is required.

The specification of which data is to be fetched and where it is to be stored is contained in the call.

10.4.1.1 Example FORTRAN Call for Get Raw Data

```
return_status =  CM50_SPGRAW
                 (%REF(id_block),
                  value_loc,
                  priority,
                  value_status,
                  cg_port_num)
```

10.4.1.2 Parameter Definitions for Get Raw Data

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `value_status` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (declare as `RECORD/ID_BLOCK_STRUCT/`), that contains the internal ID data block obtained by a previous Convert External to Internal ID request. When the data is of array type, the conversion returns the array size in word 7 of the ID block. Thus, if you wish to get less than the entire array you can change the parameter qualifier in the seventh word of the ID block to be smaller than the actual array size. Do not change any other words in the ID block. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

value_loc—The name of a Real array where the values are to be stored. The id_block should identify the value type as 7 (Real array).

priority—The name of an INTEGER*2 that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

value_status—The name of an INTEGER*2 where point-related status information is to be stored. This value is meaningful only when the return_status value indicates normal, complete with errors, or array-size error. See Appendix A.1 for a listing of Data Access error/status codes. Since `val_typ` is 7 (a Real array), `value_status` refers to status of the whole array.

cg_port_num—The name of an INTEGER*2 identifying the CG (1-4) to be accessed.

10.4.2 Store Raw Data Interface

This function stores data to a single point in the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the Convert External to Internal ID interface, see 11.2.1) is required.

The specification of where the data is found and where it is to be stored is contained in the call.

To use this function the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

10.4.2.1 Example FORTRAN Call for Store Raw Data

```
return_status =  CM50_SPSRAW
                 (%REF(id_block),
                  value_loc,
                  priority,
                  store_code,
                  value_status,
                  cg_port_num)
```

10.4.2.2 Parameter Definitions for Store Raw Data

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `value_status` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (declare as `RECORD/ID_BLOCK_STRUCT/`), that contains the internal ID data block obtained by a previous Convert External to Internal ID call. Do not change any words in the ID block. If the array size is changed, the array is not stored and the return_status value is 5 with a value_status that indicates an invalid array size. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

value_loc—The name of a Real array that contains the value or values to be stored. The id_block should identify the value type as 7 (Real array).

priority—The name of an INTEGER*2 that contains the requested data-access priority:

1 = High priority (provided for control operations)
2 = Low priority (provided for noncontrol operations)

store_code—The name of an INTEGER*2 that contains a code that allows the substitution of a bad value representation in place of the provided value(s):

0 = Store the data value(s) provided
1 = Store the bad value representation (NaN) instead

value_status—The name of an INTEGER*2 where point-related status information is to be stored. This value is meaningful only when the return_status value indicates normal or complete with errors. See Appendix A.1 for a listing of Data-Access error/status codes. Since the `val_typ` is 7 (a Real array), `value_status` refers to status of the whole array.

cg_port_num—The name of an INTEGER*2 identifying the CG (1-4) to be accessed.

10.4.3 Convert Raw Data

This function converts the elements of a Real array from LCN format to VAX format.

10.4.3.1 Example FORTRAN Call for Convert Raw Data

```
return_status =  CM50_SPCRAW
                 (%REF(id_block),
                  raw_val_loc,
                  vax_val_loc,
                  value_type,
                  convert_status)
```

10.4.3.2 Parameter Definitions for Convert Raw Data

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `convert_status` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (declare as `RECORD/ID_BLOCK_STRUCT/`), that contains the internal ID data block obtained by a previous Convert External to Internal ID call. Do not change any words in the ID block. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

raw_val_loc—The name of a Real array that contains previously obtained raw values that are to be converted from LCN format.

vax_val_loc—The name of a Real array to contain the converted values.

value_type—The name of an INTEGER*2 value that must =7 (Real array).

convert_status—The name of an INTEGER*2 array where the request-completion status for each data array element is to be stored. Value meanings are

0 = Normal return; this element was converted successfully
1 = Unable to convert this element to VAX format
2 = Bad value substitution was done on this element

10.5 HISTORY DATA TRANSFERS

The interface routines in this group get previously stored averages or 1-minute snapshot data from a History Module on the LCN. The data may be requested using a DDT, Internal Data Block or the internal address of a single tag. The History calls provide for concurrent Get History requests by up-to-four application programs. A fifth request is rejected with a queue-full status return.

10.5.1 Selecting Records from the History Module

The History Module uses a specialized set of circular files to hold historized values collected from data points on the LCN. Effective use of the CM50S history functions requires an understanding of data organization on the History Module.

10.5.1.1 Relative and Absolute Time References

The History Module may be searched using either Relative or Absolute time references. Relative references request data based on a number of records offset from the current value. Absolute Time reference request data for all records whose timestamps fall within a specified Date/Time interval.

For **Absolute Time** references, the Begin Date/Time specifies the timestamp of the most recent value to be retrieved and the End Date/Time specifies the timestamp of the oldest value to be retrieved. If a seasonal time change has occurred during a specified Absolute History interval, the number of samples returned can differ from the expected number of samples. For example, if it is desired to obtain a day's worth of hourly averages (24) and a forward time change of one hour has occurred, 23 samples are returned. If the time change is in the backward direction, 25 samples are returned.

Relative requests are based on beginning and ending offsets which are counts of records back from the current time. The direction of search can be either forward (oldest to newest data) or backwards (newest to oldest data); however, a forward search requires at least twice as long to execute. To execute a backward search, set the starting offset value less-than or equal-to the ending offset value. The number of samples returned is calculated as the positive difference between the starting offset and the ending offset plus one. If this difference exceeds 262, the request is truncated at 262 samples. The number of samples returned by a Relative History request is immune to time changes.

Offset values less than one have special meanings. When the starting or ending offset value is zero (i.e., current LCN time) in the case of averages, the first sample returned is the current running average for the period. A starting offset of -1 has special meaning in the cases of snapshots and user averages. In those cases only, LCN time is rounded to the beginning of the last hour. This permits an ACP to be sure of obtaining the last full hour of snapshots or user averages. In calculating the number of samples returned, a -1 is treated as an offset of 0 and its number of samples and direction of search follows those rules. An ending offset of -1 for snapshots and user averages means the search direction is forward and the ending time is on the hour starting "n" units back from current time.

The following table summarizes results of combinations of starting and ending offsets for Relative History requests with numbers of samples returned and reasons for zero sample returns.

History Type	Starting Offset	Ending Offset	Number of Samples	Direction of Search	Partial First Sample for Averages?
any	0	0	1	Backward	yes
any	1	1	1	Backward	no
any	2	3	2	Backward	no
any	3	2	2	Forward	no
any	0	300	262	Backward	yes
0,5	3	-1	4	Forward	no
1 to 4	3	-1	0	Error, end offset invalid	
0,5	-1	3	4	Backward	no
0,5	-1	-3	0	Error, end offset invalid	
1 to 4	-1	-3	0	Error, begin/end offset invalid	

10.5.1.2 Number of Values Retrieved in a Single Call

The number of values that can be obtained from the History Module for each point is limited both by the size of the buffer used to transfer the values and by the History type. The maximum number of values for monthly averages is 12, and for shift averages is 21. The maximum for user averages is configuration dependent, but will not exceed the number of values shown below for hourly averages. The other maximums are shown in the following table.

Number of Points in DDT or List	Maximum Snapshots	Maximum Hourly Averages	Maximum Daily Averages
1-3	262	168	31
4	262	149	31
5	238	119	31
6	198	99	31
7	170	85	31
8	149	74	31
9	132	66	31
10	119	59	31
11	108	54	31
12	99	49	31
13	91	45	31
14	85	42	31
15	79	39	31
16	74	37	31
17	69	34	31
18	66	33	31
19	62	31	31
20	59	29	29
21	56	28	28
22	53	26	27
23	51	25	25
24	49	24	24

10.5.2 Get History Snapshots (Relative Time)

These routines are used to fetch history snapshots from the HM, using a relative offset from current LCN time.

10.5.2.1 Example FORTRAN Calls for Get History Snapshots (Relative Time)

for standard 1-minute snapshots:

```
return_status =  CM50_DDTHIS_SNAP
                 (%REF(ddt_name),
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_offset,
                  end_offset)
```

for fast (5, 10 or 20 second) snapshots:

```
return_status =  CM50_DDTHIS_FAST
                 (%REF(ddt_name),
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_offset,
                  end_offset)
```

for Multi-Point Lists (instead of DDT):

```
return_status =  CM50_MPL_SNAP
                 (mpl_name,
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_offset,
                  end_offset,
                  cg_port_num)
```

for a single data point.parameter:

```
return_status =  CM50_PTHIS_SNAP
                  id_block,
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_offset,
                  end_offset,
                  cg_port_num)
```

10.5.2.2 Parameter Definitions for Get History Snapshots (Relative Time)

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a CHARACTER*9 variable that contains the ASCII name of the DDT to be used.

mpl_name—The name of a Multi-Point List structure defining the data to be retrieved. This should be declared as RECORD /CM50_IDB_REC/.

id_block—The name of a 16-byte variable containing the internal ID for an LCN tag. (declare as `RECORD /ID_BLOCK_STRUCT/`). This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

sample_rate —The name of an INTEGER*2 identifying the number of snapshots to be returned for each minute. This value does not have to match the rate at which snapshots are historized. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

Note: Retrieval of more than 1 snapshot per minute is only supported by LCN release 400.

number_of_values—The name of an INTEGER*2 that specifies the maximum number of history items (1..262) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between `begin_offset` and `end_offset`, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with status_table entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points will be lost if the `number_of_values` times the number of points is greater than 1197.

real_values_array—The name of a REAL array where the history data is to be stored (with a dimension of at least `number_of_values` times the number of points listed in the DDT). A Get Snapshot History call can return up to 1197 values.

status_table—The name of an INTEGER*2 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) to contain the value status for each returned snapshot. If the return_status value is CM50_HIS_PART (complete with errors), then for any point that could not be accessed, the first status_table entry is the Data Access Error Code (see Appendix A.1) for that point. Otherwise, each status_table entry is one of the value status codes for the corresponding real_values_array entry:

0 = Normal Data: value returned is analog (real) data
1 = Nonstandard: not applicable
2 = Digital Value: value returned is the Real equivalent of an ordinal value for a self-defined enumeration
3-4 = not used
5 = Time Change: a time change occurred and data for one minute is missing; value field contains NaN
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: History collection was not enabled; value field contains NaN
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: not applicable
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX:

CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an INTEGER*4 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) that will contain the time stamp in seconds for each returned snapshot. See heading 11.3.3 for time-stamp conversion from internal LCN format to external format.

begin_offset—The name of an INTEGER*2 that indicates a relative offset in minutes from current LCN time that represents the starting period for the history to be fetched.

end_offset—The name of an INTEGER*2 that indicates a relative offset in minutes from current LCN time representing the ending period for the history to be fetched.

cg_port_num—The name of an INTEGER*2 (with a value of 1-4) identifying the CG to be accessed.

10.5.3 Get History Snapshots (Absolute Times)

These routines are used to fetch history snapshots from the HM, using absolute begin and end times. Separate calls are provided for snapshot and averages histories.

If a seasonal time change has occurred during a specified Absolute History interval, the number of samples returned can differ from the expected number of samples. For example, if it is desired to obtain a day's worth of hourly averages (24) and a forward time change of one hour has occurred, 23 samples are returned. If the time change is in the backward direction, 25 samples are returned.

10.5.3.1 Example FORTRAN call for Get History Snapshots (Absolute Times)

for standard 1-minute snapshots:

```
return_status =  CM50_DDTHIS_SNAPT
                 (%REF(ddt_name),
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  %REF(begin_date_time),
                  %REF(end_date_time) )
```

for fast (5, 10 or 20 second) snapshots:

```
return_status =  CM50_DDTHIS_FASTT
                 (%REF(ddt_name),
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  %REF(begin_date_time),
                  %REF(end_date_time) )
```

for Multi-Point Lists (instead of DDT):

```
return_status =  CM50_MPL_SNAPT
                 (mpl_name,
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  %REF(begin_date_time),
                  %REF(end_date_time),
                  cg_port_num)
```

for a single data point.parameter:

```
return_status =  CM50_PTHIS_SNAPT
                 (id_block,
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  %REF(begin_date_time),
                  %REF(end_date_time),
                  cg_port_num)
```

10.5.3.2 Parameter Definitions for Get History Snapshots (Absolute Times)

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a CHARACTER*9 variable that contains the ASCII name of the DDT to be used.

mpl_name—The name of a Multi-Point List structure defining the data to be retrieved. This should be declared as a RECORD /CM50_IDB_REC/.

id_block—The name of a 16-byte variable containing the internal ID for an LCN tag. (declare as `RECORD /ID_BLOCK_STRUCT/`). This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

sample_rate —The name of an INTEGER*2 identifying the number of snapshots to be returned for each minute. This value does not have to match the rate at which snapshots are historized. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

Note: Retrieval of more than 1 snapshot per minute is only supported by LCN release 400.

number_of_values—The name of an INTEGER*2 that specifies the maximum number of history items (1..261) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between begin and end times, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with `status_table` entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some points are lost if the `number_of_values` times (1 + the number of points) is greater than 1197.

real_values_array—The name of a REAL array where the history data is to be stored (at least `number_of_values` times the number of points listed in the DDT). A Get Snapshot History call can return a maximum of 1197 values.

status_table—The name of an INTEGER*2 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) to contain the value type for each returned snapshot. If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: value returned is analog (real) data
1 = Nonstandard: not applicable
2 = Digital Value: value returned is the Real equivalent of an ordinal value for a self-defined enumeration
3-4 = not used
5 = Time Change: a time change occurred and data for one minute is missing; value field contains NaN
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: History collection was not enabled; value field contains NaN
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: not applicable
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX:

CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an INTEGER*4 array (with a length of at least `number_of_values` times the number of points specified in the DDT) that will contain the time stamp in seconds for each returned snapshot. See heading 11.3.3 for time-stamp conversion from internal LCN format to external format.

begin_date_time—The name of a CHARACTER*14 variable in the format MM/DD/YYΔHH:MM (where Δ indicates a blank character) specifying the date and time for the most recent record to be fetched from the History Module.

The value entered for this parameter should be some time after the desired time period begins, but before the next time period's collection time. For example, begin_date_time for an 11:00 hourly average should be set to any time from 11:01 to 11:59.

end_date_time—The name of a CHARACTER*14 variable in the format MM/DD/YYΔHH:MM, specifying the date and time for the oldest record to be fetched from the History Module. The end_date_time must be earlier than begin_date_time.

The value entered for this parameter should be some time after the previous period's collection, but before the desired time period begins. For example, to get an hourly average at 11:00, end_date_time should be set any time between 10:01 and 10:59.

cg_port_num—The name of an INTEGER*2 (with a value of 1-4) identifying the CG to be accessed.

10.5.4 Get History Averages (Relative Times)

These calls return the average, minimum and maximum values of a point for specified time periods.

10.5.4.1 Example FORTRAN call for Get History Averages (Relative Times)

```
return_status =  CM50_DDTHIS_AVER
                 (%REF(ddt_name),
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset,
                  history_type)
```

for Multi-Point Lists (instead of DDT):

```
return_status =  CM50_MPLHIS_AVER
                 (mpl_name,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset,
                  history_type,
                  cg_port_num)
```

for a single data point.parameter:

```
return_status =  CM50_PTHIS_AVER
                 (id_block,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset,
                  history_type,
                  cg_port_num)
```

10.5.4.2 Parameter Definitions for Get History Averages (Relative Times)

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a CHARACTER*9 variable that contains the ASCII name of the DDT to be used.

mpl_name—The name of a Multi-Point List structure defining the data to be retrieved. This should be declared as RECORD /CM50_IDB_REC/.

id_block—The name of a 16-byte variable containing the internal ID for an LCN tag. (declare as `RECORD /ID_BLOCK_STRUCT/`). This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of an INTEGER*2 that specifies the maximum number of history items (1..262) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between `begin_offset` and `end_offset`, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with `status_table` entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times the number of points is greater than 598.

real_values_array—The name of a REAL array where the history data is to be stored (at least `number_of_values` times the number of points listed in the DDT). A Get Averages History call can return a maximum of 598 values.

status_table—The name of an INTEGER*2 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) to contain the value type for each returned snapshot. If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX:
CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an INTEGER*4 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) that will contain the time stamp in seconds for each returned average. See heading 11.3.3 for time-stamp conversion from internal LCN format to external format.

max_array—The name of a REAL array (of length at least `number_of_values` times the number of points specified in the DDT) that will contain the maximum process value recorded in the averaged period. Due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of a REAL array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) that will contain the minimum process value recorded in the averaged period. Due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an INTEGER*2 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) that will contain the number of samples used in calculating each returned average value. Note that monthly averages can contain up to 44,640 samples, and any 16-bit number larger than 32,767 appears as a negative value in FORTRAN. To see accurate values for monthly samples, use the following logic with an INTEGER*4 variable named TEMP:

```
IF (num_samples_array(i) .LT. 0) THEN
        TEMP = 65536 + num_samples_array(i)
ELSE
        TEMP = num_samples_array(i)
ENDIF
```

begin_offset—The name of an INTEGER*2 that indicates a relative offset from current LCN time that represents the first history record to be fetched.

end_offset—The name of an INTEGER*2 that indicates a relative offset from the current LCN time representing the last history record to be fetched.

history_type—The name of an INTEGER*2 that contains the number specifying the type of average requested. The available types and maximum number of records on the History Module for each are:

1 = Hourly	(168	records)
2 = Shift	(21	records)
3 = Daily	(31	records)
4 = Monthly	(12	records)
5 = User	(configuration dependent)	

cg_port_num—The name of an INTEGER*2 (with a value of 1-4) identifying the CG to be accessed.

10.5.5 Get History Averages (Absolute Times)

These calls return the average, minimum and maximum values of a point for specified time periods.

10.5.5.1 Example FORTRAN call for Get History Averages (Absolute Times)

```
return_status =  CM50_DDTHIS_AVERT
                 (%REF(ddt_name),
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  %REF(begin_date_time),
                  %REF(end_date_time),
                  history_type)
```

for Multi-Point Lists (instead of DDT):

```
return_status =  CM50_MPLHIS_AVERT
                 (MPL,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  %REF(begin_date_time),
                  %REF(end_date_time),
                  history_type,
                  cg_port_num)
```

for single point requests:

```
return_status =  CM50_PTHIS_AVERT
                 (id_block,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  %REF(begin_date_time),
                  %REF(end_date_time),
                  history_type,
                  cg_port_num)
```

10.5.5.2 Parameter Definitions for Get History Averages (Absolute Times)

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-CHARACTER* variable that contains the ASCII name of the DDT to be used.

mpl —The name of a Multi-Point List structure defining the data to be retrieved. This should be declared as RECORD /CM50_IDB_REC/.

id_block—The name of a 16-byte variable containing the internal ID for an LCN tag. (declare as `RECORD /ID_BLOCK_STRUCT/`). This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of an INTEGER*2 that specifies the maximum number of history items (1..261) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between begin and end times, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with status_table entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times (1 + the number of points) is greater than 598.

real_values_array—The name of a REAL array where the history data is to be stored (at least `number_of_values` times the number of points listed in the DDT). A Get Averages History call can return a maximum of 598 values.

status_table—The name of an INTEGER*2 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) to contain the value type for each returned snapshot. If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX

CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38

CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38

CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38

CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38

CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an INTEGER*4 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) to receive the time stamp in seconds for each returned average. See heading 11.3.3 for time-stamp conversion from internal LCN format to external format.

max_array—The name of a REAL array (of length at least `number_of_values` times the number of points specified in the DDT) that will contain the maximum process value recorded in the averaged period. **Note:** Due to the data compression algorithm on the History module, there may be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of a REAL array (of length at least `number_of_values` times the number of points specified in the DDT) that will contain the minimum process value recorded in the averaged period. **Note:** Due to the data compression algorithm on the History module, there may be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an INTEGER*2 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) that will contain the number of samples used in calculating each returned average value. Note that monthly averages can contain up to 44,640 samples, and any 16-bit number larger than 32,767 appears as a negative value in FORTRAN. To see accurate values for monthly samples, use the following logic with an INTEGER*4 variable named TEMP:

```
IF (num_samples_array(i) .LT. 0) THEN
        TEMP = 65536 + num_samples_array(i)
ELSE
        TEMP = num_samples_array(i)
ENDIF
```

begin_date_time—The name of a CHARACTER*14 variable in the format MM/DD/YYΔHH:MM (where Δ indicates a blank character) specifying the date and time for the most recent record to be fetched from the History Module.

The value entered for this parameter should be some time after the desired time period begins, but before the next time period's collection time. For example, begin_date_time for an 11:00 hourly average should be set to any time from 11:01 to 11:59.

end_date_time—The name of a CHARACTER*14 variable in the format MM/DD/YYΔHH:MM, specifying the date and time for the oldest record to be fetched from the History Module. The end_date_time must be earlier than the begin_date_time.

The value entered for this parameter should be some time after the previous period's collection, but before the desired time period begins. For example, to get an hourly average at 11:00, end_date_time should be set any time between 10:01 and 10:59.

history_type—The name of an INTEGER*2 that contains the number specifying the type of average requested. The available types and maximum time retained on the History Module for each are:

1 = Hourly (7 days)
2 = Shift (7 days)
3 = Daily (31 days)
4 = Monthly (1 year)
5 = User (8 hours to 7 days, depending on configuration)

cg_port_num—The name of an INTEGER*2 (with a value of 1-4) identifying the CG to be accessed.

10.5.6 Get Monthly Averages (Relative Times)

When a point is historized more often than once per minute, it is possible for the number of samples taken during a month to exceed the capacity of a 16-bit integer. This call provides a 32-bit integer count of the number of samples in a monthly average using relative time.

NOTE

Retrieval of monthly averages using this call is only supported by LCN release 400 or later.

10.5.6.1 Example FORTRAN call for Get Monthly Averages (Relative Times)

```
return_status =  CM50_DDTHIS_MNTH
                 (%REF(ddt_name),
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset)
```

for Multi-Point Lists (instead of DDT):

```
return_status =  CM50_MPLHIS_MNTH
                 (mpl_name,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset,
                  cg_port_num)
```

for a single data point.parameter:

```
return_status =  CM50_PTHIS_MNTH
                 (id_block,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset,
                  cg_port_num)
```

10.5.6.2 Parameter Definitions for Get Monthly Averages (Relative Times)

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a CHARACTER*9 variable that contains the ASCII name of the DDT to be used.

mpl_name—The name of a Multi-Point List structure defining the data to be retrieved. This should be declared as RECORD /CM50_IDB_REC/.

id_block—The name of a 16-byte variable containing the internal ID for an LCN tag (declare as `RECORD /ID_BLOCK_STRUCT/`). This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of an INTEGER*2 that specifies the maximum number of history items (1..262) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between `begin_offset` and `end_offset`, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with `status_table` entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times the number of points is greater than 598.

real_values_array—The name of a REAL array where the history data is to be stored (at least `number_of_values` times the number of points listed in the DDT). A Get Averages History call can return a maximum of 598 values.

status_table—The name of an INTEGER*2 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) to contain the value type for each returned snapshot. If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX:
CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an INTEGER*4 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) that will contain the time stamp in seconds for each returned average. See heading 11.3.3 for time-stamp conversion from internal LCN format to external format.

max_array—The name of a REAL array (of length at least `number_of_values` times the number of points specified in the DDT) that will contain the maximum process value recorded in the averaged period. Due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of a REAL array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) that will contain the minimum process value recorded in the averaged period. Due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an INTEGER*4 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) that will contain the number of samples used in calculating each returned average.

begin_offset—The name of an INTEGER*2 that indicates a relative offset from current LCN time that represents the first history record to be fetched.

end_offset—The name of an INTEGER*2 that indicates a relative offset from the current LCN time representing the last history record to be fetched.

cg_port_num—The name of an INTEGER*2 (with a value of 1-4) identifying the CG.

10.5.7 Get Monthly Averages (Absolute Times)

When a point is historized more often than once per minute, it is possible for the number of samples taken during a month to exceed the capacity of a 16-bit integer. This call provides a 32-bit integer count of the number of samples in a monthly average using absolute time.

NOTE

Retrieval of monthly averages using this call is only supported by LCN release 400 or later.

10.5.7.1 Example FORTRAN call for Get Monthly Averages (Absolute Times)

```
return_status =  CM50_DDTHIS_MNTHT
                 (%REF(ddt_name),
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  %REF(begin_date_time),
                  %REF(end_date_time))
```

for Multi-Point Lists (instead of DDT):

```
return_status =  CM50_MPLHIS_MNTHT
                 (MPL,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  %REF(begin_date_time),
                  %REF(end_date_time),
                  cg_port_num)
```

for single point requests:

```
return_status =  CM50_PTHIS_MNTHT
                 (id_block,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  %REF(begin_date_time),
                  %REF(end_date_time),
                  cg_port_num)
```

10.5.7.2 Parameter Definitions for Get Monthly Averages (Absolute Times)

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-CHARACTER* variable that contains the ASCII name of the DDT to be used.

mpl —The name of a Multi-Point List structure defining the data to be retrieved. This should be declared as RECORD /CM50_IDB_REC/.

id_block—The name of a 16-byte variable containing the internal ID for an LCN tag. (declare as `RECORD /ID_BLOCK_STRUCT/`). This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of an INTEGER*2 that specifies the maximum number of history items (1..261) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between begin and end times, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with status_table entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times (1 + the number of points) is greater than 598.

real_values_array—The name of a REAL array where the history data is to be stored (at least `number_of_values` times the number of points listed in the DDT). A Get Averages History call can return a maximum of 598 values.

status_table—The name of an INTEGER*2 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) to contain the value type for each returned snapshot. If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX
CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an INTEGER*4 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) to receive the time stamp in seconds for each returned average. See heading 11.3.3 for time-stamp conversion from internal LCN format to external format.

max_array—The name of a REAL array (of length at least `number_of_values` times the number of points specified in the DDT) that will contain the maximum process value recorded in the averaged period. **Note:** Due to the data compression algorithm on the History module, there may be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of a REAL array (of length at least `number_of_values` times the number of points specified in the DDT) that will contain the minimum process value recorded in the averaged period. **Note:** Due to the data compression algorithm on the History module, there may be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an INTEGER*2 array (with a dimension of at least `number_of_values` times the number of points specified in the DDT) that will contain the number of samples used in calculating each returned average value.

begin_date_time—The name of a CHARACTER*14 variable in the format MM/DD/YYΔHH:MM (where Δ indicates a blank character) specifying the date and time for the most recent record to be fetched from the History Module.

The value entered for this parameter should be some time after the desired time period begins, but before the next time period's collection time. For example, begin_date_time for an 11:00 hourly average should be set to any time from 11:01 to 11:59.

end_date_time—The name of a CHARACTER*14 variable in the format MM/DD/YYΔHH:MM, specifying the date and time for the oldest record to be fetched from the History Module. The end_date_time must be earlier than the begin_date_time.

The value entered for this parameter should be some time after the previous period's collection, but before the desired time period begins. For example, to get an hourly average at 11:00, end_date_time should be set any time between 10:01 and 10:59.

cg_port_num—The name of an INTEGER*2 (with a value of 1-4) identifying the CG to be accessed.

10.5.8 Historization Sampling Rate Queries

These functions query the LCN and return the current Historization Sampling Rate (number of snapshots recorded each minute) for a point or set of points.

NOTE

Retrieval of sampling rates using this call is only supported by LCN release 400 or later.

10.5.8.1 Example FORTRAN calls for Query Sampling Rate

For Points referenced in a History DDT:

```
return_status =  CM50_DDTHIS_RATE
                 (%REF(ddt_name),
                  history_rate_array,
                  status_table)
```

For a List of Internal Point ids:

```
return_status =  CM50_MPLHIS_RATE
                 (mpl_name,
                  history_rate_array,
                  status_table,
                  cg_port_number)
```

For a Point addressed by its internal id:

```
return_status =  CM50_PTHIS_RATE
                 (id_block,
                  history_rate,
                  cg_port_number)
```

For a Point addressed by its internal id:

```
return_status =  CM50_TAGHIS_RATE
                 (%REF(tagname),
                  history_rate,
                  cg_port_number)
```

10.5.8.2 Parameter Definitions for History Sampling Rate Queries

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a CHARACTER*9 variable that contains the ASCII name of the DDT to be used.

mpl_name—The name of a Multi-Point List structure defining the data to be retrieved. This should be declared as a RECORD /CM50_IDB_REC/.

id_block—The name of a 16-byte variable containing the internal ID for an LCN tag (declare as `RECORD /ID_BLOCK_STRUCT/`). This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

tagname—The name of a CHARACTER*40 variable that contains an LCN tagname in the form "point.parameter(index)", where the "(index)" is used only to identify elements of an array.

history_rate —The name of an INTEGER*2 identifying the number of snapshots to collected each minute. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

history_rate_array —The name of an INTEGER*2 array (with a dimension of at least the number of points in the DDT or list) identifying the number of snapshots collected each minute. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

status_table—The name of an INTEGER*2 array (with a dimension of at least the number of points specified in the DDT or list) to contain the data access code for each point (See appendix A.1).

10.6 TEXT MESSAGE TRANSFERS

The two interface routines in this group are used to send and receive character-string messages over the LCN.

10.6.1 Get Message Interface

This routine is used to fetch a character-string message held in a buffer by this program's ACIDP. The message presence is determined as the result of a Get ACP Status request.

10.6.1.1 Example FORTRAN Call for Get Message

```
return_status =  CM50_GETMSG
                 (%REF(msg),
                  msg_len)
```

10.6.1.2 Parameter Definitions for Get Message

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. There are three non-normal return_status values for this call that indicate the need for additional processing:

215000521	CM50_MSG_TRUNC	Received message was truncated
215000561	CM50_MSG_QUE	Message was received and another one is queued
215000571	CM50_MSG_QUET	Received message was truncated & another one is queued

msg—The name of a CHARACTER*120 variable where the message is to be stored.

msg_len—The name of an INTEGER*2 that specifies the maximum number of characters to accept (120-character limit).

10.6.2 Send Message Interface

This routine is used to send a message to all operator stations assigned to the same unit as this program's ACIDP. A request to wait for operator confirmation is optional. If operator confirmation is requested, execution of the requesting program is suspended until either the confirmation occurs, or until its specified wait time expires. The requesting program receives an indication of whether confirmation or a time out occurs.

10.6.2.1 Example FORTRAN Call for Send Message

```
return_status =  CM50_STOREMSG
                 (%REF(msg),
                  msg_len,
                  confirm,
                  timeout,
                  dest)
```

10.6.2.2 Parameter Definitions for Send Message

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

msg—The name of a CHARACTER*120 (or less)variable that contains the message to be sent.

msg_len—The name of an INTEGER*2 that specifies the number of characters to be transmitted. The maximum number of characters depends on message destination: 60 for CRT displays and 72 for printing. Over-length messages are truncated. All messages are archived if the HM is so configured.

confirm—The name of a Boolean INTEGER*2 (1 = TRUE and 0 = FALSE) that specifies whether or not a message confirmation is required. Note that this parameter is treated as FALSE if the message destination is printer only.

timeout—The name of an INTEGER*2 that specifies the number of seconds (0 to 3600) the system is to wait for confirmation before returning control to the requesting program with a "no confirm" return_status. (Allow for a built-in time lag of up-to-10 seconds.) The Wait Time parameter is ignored if the Confirm parameter is set to OFF or the message destination is printer only.

dest—The name of an INTEGER*2 that specifies where the message is to be sent, as follows:

0 – CRT only
1 – Printer only
2 – Both

10.6.2.3 Event-Initiated Reports

Two types of Event-Initiated Reports can be invoked by specially formatted messages from an ACP or an Indirect Control Program to the Area Universal Stations:

- Logs, reports journals, and trends configured in the Area database
- Event History reports

Details of message requirements are given in Section 30 of the *Engineer's Reference Manual* located in the *Implementation/Startup & Reconfiguration* - 2 binder.

PROGRAM CONTROL AND SUPPORT (FORTRAN)
Section 11

This section discusses program interfaces that support and control Advanced Control programs

11.1 ACP EXECUTION SUPPORT

These interface routines affect the orderly execution and termination of application programs.

11.1.1 ACP Initialization Interface

This routine (or the vintage ACPTRP procedure) **must** be the first executable statement in each ACP but is optional for DAPs and Indirect Control Programs. It establishes a termination handler and ensures proper ACP table setup. Failure to invoke this interface routine as the first statement of an ACP may not appear to cause immediate problems, but will result in improper termination handling. The termination status is not reported to the CG, and the ACP appears to both the CM50S and the CG to still be in the RUN state even though the process has terminated.

The call to CM50_SET_ACP also establishes a system lock that allows the program to be terminated cleanly if CM50S is shut down. Therefore, it is advisable to include this call in every program that is mapped to the CM50S shareable image.

11.1.1.1 Example FORTRAN Call for ACP Initialization

```
return_status = CM50_SET_ACP
                      (reset)
```

11.1.1.2 Parameter Definitions for ACP Initialization

return_status—The name of an INTEGER*4 to receive the overall return status of the function. This function always returns as a success (`return_status` = 1).

reset—The name of an INTEGER*2 that specifies the reaction of the trap handler to an abort. If the ACP is aborted for any reason, the Abort code is recorded in the CG/PLNM database and the ACP Status table. If the value of **reset** is 1, then the execution status of the ACP is reset to OFF/DELAY regardless of how the program terminated. For any other value of **reset**, the execution status of the ACP becomes OFF/DELAY only after normal termination and is set to ABORT after an abnormal program termination.

11.1.2 Get ACP Status Interface

This routine fetches a set of parameters that enables the requesting ACP to determine why the system has turned it on and what special processing may be required at this time. It should be used during both the "setup" and "cleanup" program stages each time an ACP runs. After servicing this request, the interface routine resets its copy of these values in preparation for any subsequent ACP turn on.

NOTE

GETSTS is one of the few CM50S user-interface routines that is not implemented as a function. It is called as a FORTRAN subroutine.

11.1.2.1 Example FORTRAN Call for Get ACP Status

```
CALL GETSTS
     (take_i_p,
      ps_msg,
      demand,
      procspec,
      scheduled,
      upper_level)
```

11.1.2.2 Parameter Definitions for Get ACP Status

take_i_p—The name of a Boolean INTEGER*2 (-1 = TRUE and 0 = FALSE) that returns TRUE the first time this program is turned on by the CG, following an initialization event (see heading 4.4.1). `take_i_p` should be ignored when `upper_level` is TRUE.

ps_msg—The name of a Boolean INTEGER*2 (-1 = TRUE and 0 = FALSE) that returns TRUE if a message for the program is waiting at the CG.

demand—The name of a Boolean INTEGER*2 (-1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on as the result of a process operator request.

procspec—The name of a Boolean INTEGER*2 (-1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on as the result of a process special to its ACIDP from an HG, AM, or another ACP.

scheduled—The name of a Boolean INTEGER*2 (-1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on by periodic or cyclic scheduling.

upper_level—The name of a Boolean INTEGER*2 (-1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on by the VAX.

11.1.3 ACP Delay Interface

This routine suspends execution of the calling program for a specified number of seconds. Program execution resumes at the statement following the delay call.

11.1.3.1 Example FORTRAN Call for ACP Delay

```
sleep = CM50_ACPDELAY
        (delay_time)
```

11.1.3.2 Parameter Definitions for ACP Delay

sleep—The name of a Boolean INTEGER*2 (1 = TRUE and 0 = FALSE) that contains the overall return status of the function call. The value will be FALSE when the call has been rejected because of an invalid delay time value.

delay_time—The name of an INTEGER*2 that contains the length of time (1 to 60 seconds) that the requesting program is to be suspended.

11.1.4 ACP Hibernate Interface

This routine suspends execution of the calling ACP (through a VMS SYS$HIBER request) until the next turn on request. The program and associated data remain in memory during hibernation, in effect making it memory-resident. Program execution resumes at the statement following the CM50_HIBER call.

11.1.4.1 Example FORTRAN Call for ACP Hibernate

```
hiber_stat = CM50_HIBER ()
```

Note: The empty argument list "( )" is required when calling the function from FORTRAN.

11.1.4.2 Parameter Definitions for ACP Hibernate

hiber_stat—The name of an INTEGER*4 to receive the overall return status of the function call. This value should always = 1 (SS$_NORMAL). Any other value indicates a fatal error. The program should call the GETSTS routine (see heading 11.1.2) to determine how the Wake call was issued.

11.1.5 ACP Termination Interface

This routine terminates the execution of the calling ACP. It must be used as the last operating statement of each ACP but is optional for DAPs and Indirect Control Programs.

For ACPs, this call stores a termination-status code in the associated ACIDP's ABORTCOD parameter. The termination code can be viewed at a Universal Station (see the definitions for ABORTCOD and EXECSTAT at heading 4.4.1), but in a revised form. The integer value assigned here is translated into two hexadecimal digits (00 to FF) and appended to the major code of 'EA'. Thus, an ACP-assigned abnormal termination code of 15 appears at the Universal Station display as 'EA0F'.

If an ACP is aborted by the VMS operating system, an abort code of 'VMSF' is stored in its ACIDP's ABORTCOD.

The execution state of an ACIDP can be changed from ABORT to normal by operator demand through a Universal Station or by invoking the ACP Operation screen's Deactivate/Terminate function. See heading 5.8 for abort recovery details.

NOTE

PRGTRM is one of the few user-interface routines that is not implemented as a function. It is called as a FORTRAN subroutine.

11.1.5.1 Example FORTRAN Call for ACP Termination

```
CALL   PRGTRM
       (terminate_code)
```

11.1.5.2 Parameter Definitions for ACP Termination

terminate_code—The name of an INTEGER*4 that must contain zero or a positive value (1 to 255). Zero value indicates normal termination. Nonzero values are user-specified codes for nonnormal termination (abort). Note that if you provide a value outside the valid range, ABORTCOD will contain EAΔΔ (where.Δ represents a blank).

11.2 ENTITY NAME CONVERSIONS

The interface routines in this group convert ASCII names of LCN points and parameters into their internal LCN identifiers.

NOTE

The arrays of internal point.parameter addresses need to be rebuilt and the program(s) using them need to be recompiled whenever the LCN database is changed in a significant manner, such as by the rebuild or deletion of data points referenced in the address array.

11.2.1 Convert External to Internal ID

These routines fetch the internal ID of a point.parameter for the calling program. Use of the internal ID by repetitive single-value data gets and stores reduces system overhead and provides faster return of data. The specification of which point.parameter internal ID is wanted and where it is to be stored is contained in the call.

11.2.1.1 Example FORTRAN Calls for Convert ID

Using point and parameter names as separate variables:

```
return_status =     CM50_CONV_PT
                    (%REF(entity),
                     %REF(param),
                     param_ix,
                     id_block,
                     val_typ,
                     cg_port_num)
```

When the external id is expressed as a Tag name (not separate point and parameter), use:

```
return_status =     CM50_CONV_TAG
                    (%REF(tag_name),
                     id_block,
                     val_typ,
                     cg_port_num)
```

11.2.1.2 Parameter Definitions for Convert ID

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000146 (CM50_LCN_ARRAY)—the array size specified by `param_ix` is smaller the actual array size.

215000322 (CM50_ACC_SIZE)—the array size specified by `param_ix` is larger than the actual array size.

tag_name—The name of a CHARACTER*40 variable that identifies the LCN value(s) to be stored. The tag name is formatted as "point.param (param_ix)".

entity—The name of a CHARACTER*20 variable that contains the ASCII Point ID. It should contain a point name of up to 16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter for Network Gateway routing.

param—The name of a CHARACTER*8 variable that contains the ASCII name of the parameter to be converted.

param_ix—The name of an INTEGER*2 positive value. Use of this value is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, 5, 13, 15, 17 or 19, a single value is to be accessed—This may be an element of a parameter array in which case `param_ix` identifies the element within the array. If the parameter being accessed is not an array type, `param_ix` must be zero.

When `val_typ` is 7, 8, 9, 10, 14, 16, 18 or 20, a whole array (or a subset of the array starting with the first element) is to be accessed and `param_ix` is used to specify the size of the array. If `param_ix` is smaller than the actual array size, the conversion is made; if it is larger than the actual array size, the conversion is not made. Both conditions cause non-normal `return_status` values to be returned.

id_block—The name of a 16-byte variable where the internal ID data block is to be returned (declare as `RECORD /ID_BLOCK_STRUCT/`). Save these eight values for later use in calls on this point.parameter. The ID data block contents are as follows:

Word 1—	Data type
Words 2..5—	Internal point identifier
Word 6—	Parameter subscript
Word 7—	Parameter qualifier (array size)
Word 8—	Enumeration set identifier

val_typ—The name of an INTEGER*2 that contains a number that designates value type. If the incorrect value is supplied on input, this value will be updated as an output variable. The coded values:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
6 = not used
7 = Real array
8 = Integer array
9 = Enumeration array
10 = Ordinal value of enumeration array
13 = Internal entity id
14 = Internal entity id array
15 = External entity id
16 = External entity id array
17 = Time value
18 = Time value array
19 = String value
20 = String value array

cg_port_num—The name of an INTEGER*2 (with a value of 1-4) identifying the CG to be accessed.

11.2.2 Convert List of External IDs

These routines fetch an array of internal IDs for a list of point.parameters. These calls are designed for use with the Point Array calls described in section 10.2. All of the point.parameters in each list must be of the same data type (Real, ASCII, etc.).

11.2.2.1 Example FORTRAN Call for Convert Lists

When the point & parameter names are maintained separately, use:

```
return_status =     CM50_CONV_PT_LIST
                    (%REF(entity_arr),
                     %REF(param_arr),
                     param_ix_arr,
                     number_of_values,
                     val_typ,
                     cg_port_num
                     id_block_arr,
                     return_arr)
```

When the external id is expressed as a Tag name (not separate point and parameter), use:

```
return_status =     CM50_CONV_TAG_LIST
                    (%REF(Tagname_arr),
                     number_of_values,
                     val_typ,
                     cg_port_num
                     id_block_arr,
                     return_arr)
```

11.2.2.2 Parameter Definitions for Convert Lists

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `return_arr` array entry for each returned id block must be checked for errors.

return_arr—The name of an array of up to 300 INTEGER*4 variables to receive the status of the conversion of each point or tag. See Appendix A.2 for an explanation and a listing of all assigned return code values.

tagname_arr—The name of an array of up to 300 CHARACTER*40 variables. Each variable contains the ASCII tagname of the LCN entity for which the internal ID is to be obtained. The tagnames are formatted as `Point.Parameter` or `Point.Parameter(ix)`, where (ix) is an array element index used only with array parameters.

entity_arr—The name of an array of up to 300 CHARACTER*20 variables, each containing an ASCII Point id. Each variable should contain the point name of up to 16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter for Network Gateway routing, of a point for which the internal ID is to be obtained.

param_arr—The name of an array of up to 300 CHARACTER*8 variables, each containing the ASCII parameter name of a point.parameter for which the internal ID is to be obtained.

param_ix_arr—The name of an array of up to 300 INTEGER*2 variables used as array element index values corresponding to the individual parameter names in param_arr. For each non-array parameter named in that array, the corresponding value in this array should be zero.

cg_port_num—The name of an INTEGER*2 (with a value of 1-4) identifying the CG to be accessed.

id_block_arr—The name of an array of up to 300 16-byte variables where the internal ID data blocks are to be returned (declare as array of `RECORD /ID_BLOCK_STRUCT/`). The ID data block contents are as follows:

Word 1—	Data type
Words 2..5—	Internal point identifier
Word 6—	Parameter subscript
Word 7—	Parameter qualifier (array size)
Word 8—	Enumeration set identifier

number_of values—The name of an INTEGER*2 that contains the number of points/tags in the list to be converted.

val_typ—The name of an INTEGER*2 that contains a number that designates LCN value type. This value must be supplied in the calling argument. The acceptable values are:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
13 = Internal entity id
15 = External entity id
17 = Internal Time
19 = String

11.3 VALUE CONVERSIONS

The interface routines in this group convert values from one internal format to another.

11.3.1 Valid Number Check

This routine checks a value of type "Real" to determine if it is a valid single-precision, floating-point number. Its primary purpose is to check for the "Bad Value" indicator, NaN (-0).

11.3.1.1 Example FORTRAN Call for Valid Number Check

```
value_st =  CM50_VALIDN
        (value)
```

11.3.1.2 Parameter Definitions for Valid Number Check

value_st—The name of a Boolean INTEGER*2 (1 = TRUE and 0 = FALSE) that returns TRUE if "Value" is found to be a valid floating-point number. It returns FALSE for minus zero (NaN) or other invalid bit configurations.

value—The name of a REAL variable that contains a single-precision, floating-point value that is to be checked. When `value_st` returns FALSE, the contents of `value` have been changed to 0.0.

11.3.2 Set Bad Value

This routine stores the bad value constant, NaN (-0), into the specified Real variable.

11.3.2.1 Example FORTRAN Call for Set Bad Value

```
return_status = CM50_SETBAD (var_name)
              or
CALL CM50_SETBAD (var_name)
```

11.3.2.2 Parameter Definitions for Set Bad Value

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For this function, `return_status` = 1 always.

var_name—The name of a REAL variable where the bad value constant for the LCN is to be stored. Note that attempting to perform arithmetic operations using a variable that has been set to Bad Value can cause a fatal error within the program.

11.3.3 Convert Time Values

Within the CM50 environment, Date/time variables are often maintained in a variety of formats. The following routines convert time values from any one of the following formats to any other:

abbrev.	format	use
LCN	4-byte integer	internal LCN clock, number of seconds since January 1, 1979
VAXB	8-bytes (array of two 4-byte integers)	VAX binary system clock format
VAXA	22 characters	VAX standard ASCII time display: 'dd-MON-yyyy hh:mm:ss'
ASC	18 characters	LCN standard ASCII time display 'mm/dd/yy hh:mm:ss'
EURO	18 characters	European ASCII time display 'dd/mm/yy hh:mm:ss'
ARY	12 bytes (equivalenced to six 2-byte integers)	FORTRAN integer*2 array with element: 1 = year 2 = month 3 = day 4 = hour 5 = minute 6 = second

In each routine, the first argument must be assigned the input value and the second argument is the returned converted value.

11.3.3.1 Example FORTRAN Calls to Convert Time

Convert internal LCN time to an INTEGER*2 array:

```
return_status = CM50_TIMLCN_ARY
                      (lcn,
                       ary)
```

Convert internal LCN time to an ASCII string:

```
return_status = CM50_TIMLCN_ASC
                      (lcn,
                       %REF(asc))
```

Convert internal LCN time to a European string:

```
return_status = CM50_TIMLCN_EURO
                      (lcn,
                       %REF(euro))
```

Convert internal LCN time to VAX display format:

```
return_status = CM50_TIMLCN_VAXA
                      (lcn,
                       %REF(vaxa))
```

Convert internal LCN time to VAX binary:

```
    return_status = CM50_TIMLCN_VAXB
                            (lcn,
                             vaxb)
```

Convert an INTEGER*2 array to internal LCN:

```
    return_status = CM50_TIMARY_LCN
                            (ary,
                             lcn)
```

Convert an INTEGER*2 array to an ASCII string:

```
    return_status = CM50_TIMARY_ASC
                            (ary,
                             %REF(asc))
```

Convert an INTEGER*2 array to a European string:

```
    return_status = CM50_TIMARY_EURO
                            (ary,
                             %REF(euro))
```

Convert an INTEGER*2 array to VAX display format:

```
    return_status = CM50_TIMARY_VAXA
                            (ary,
                             %REF(vaxa))
```

Convert an INTEGER*2 array to VAX binary:

```
    return_status = CM50_TIMARY_VAXB
                            (ary,
                             vaxb)
```

Convert an ASCII string to internal LCN:

```
    return_status = CM50_TIMASC_LCN
                            (%REF(asc),
                             lcn)
```

Convert an ASCII string to an INTEGER*2 array:

```
    return_status = CM50_TIMASC_ARY
                            (%REF(asc),
                             ary)
```

Convert an ASCII string to a European string:

```
    return_status = CM50_TIMASC_EURO
                            (%REF(asc),
                             %REF(euro))
```

Convert an ASCII string to VAX display format:

```
    return_status = CM50_TIMASC_VAXA
                            (%REF(asc),
                             %REF(vaxa))
```

Convert an ASCII string to VAX binary:

```
return_status = CM50_TIMASC_VAXB
                        (%REF(asc),
                         vaxb)
```

Convert a European string to internal LCN:

```
return_status = CM50_TIMEURO_LCN
                        (%REF(euro),
                         lcn)
```

Convert a European string to an INTEGER*2 array:

```
return_status = CM50_TIMEURO_ARY
                        (%REF(euro),
                         ary)
```

Convert a European string to an ASCII string:

```
return_status = CM50_TIMEURO_ASC
                        (%REF(euro),
                         %REF(asc))
```

Convert a European string to VAX display format:

```
return_status = CM50_TIMEURO_VAXA
                        (%REF(euro),
                         %REF(vaxa))
```

Convert a European string to VAX binary:

```
return_status = CM50_TIMEURO_VAXB
                        (%REF(euro),
                         vaxb)
```

Convert VAX display format to internal LCN:

```
return_status = CM50_TIMVAXA_LCN
                        (%REF(vaxa),
                         lcn)
```

Convert VAX display format to an INTEGER*2 array:

```
return_status = CM50_TIMVAXA_ARY
                        (%REF(vaxa),
                         ary)
```

Convert VAX display format to an ASCII string:

```
return_status = CM50_TIMVAXA_ASC
                        (%REF(vaxa),
                         %REF(asc))
```

Convert VAX display format to a European string:

```
return_status = CM50_TIMVAXA_EURO
                        (%REF(vaxa),
                         %REF(euro))
```

Convert VAX display format to VAX binary:

```
return_status = CM50_TIMVAXA_VAXB
                        (%REF(vaxa),
                         vaxb)
```

Convert VAX binary to internal LCN:

```
    return_status = CM50_TIMVAXB_LCN
                            (vaxb,
                             lcn)
```

Convert VAX binary to an INTEGER*2 array:

```
    return_status = CM50_TIMVAXB_ARY
                            (vaxb,
                             ary)
```

Convert VAX binary to an ASCII string:

```
    return_status = CM50_TIMVAXB_ASC
                            (vaxb,
                             %REF(asc))
```

Convert VAX binary to a European string:

```
    return_status = CM50_TIMVAXB_EURO
                            (vaxb,
                             %REF(euro))
```

Convert VAX binary to VAX display format:

```
    return_status = CM50_TIMVAXB_VAXA
                            (vaxb,
                             %REF(vaxa))
```

11.3.3.2 Parameter Definitions for Convert Time Values

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

lcn—The name of an INTEGER*4 that contains a value representing internal LCN time to the nearest second.

ary—The name of a 12-byte string (or array of 6 INTEGER*2 values) that contains a value representing a date and time.

asc—The name of a CHARACTER*18 variable representing time in the format : 'mm/dd/yy hh:mm:ss '.

euro—The name of a CHARACTER*18 variable representing time in the format : 'dd/mm/yy hh:mm:ss '.

vaxa—The name of a CHARACTER*22 variable representing time in the format: 'dd-MON-yyyy hh:mm:ss', where MON represents the first three letters (in upper case) of the English name of the month.

vaxb—The name of a 64-bit variable (array of 2 INTEGER*4s) that contains a value representing internal VAX binary time.

CM50S ADMINISTRATION (FORTRAN)
Section 12

This section discusses the programmatic calls that can be used to manage the ACPs and DDTs installed in a CM50S system.

12.1 PROGRAMMATIC INTERFACES TO ACP OPERATIONS

A programmatic interface to all ACP Operations gives users programmatic access to the same ACP functions that are available through the ACP Operations user interface. In order to use the ACP Programmatic Interface, the user should include the ACP Include files (`CM50_FLAGS_INCLUDE.FOR` and `CM50_ACP_INCLUDE.FOR`). These files define data types and routines required by the Programmatic Interface calls. The following sections discuss the ACP Programmatic Interface calls in detail.

12.1.1 Install ACP

This routine is called to install an ACP. The ACP can be installed under a different name than the executable filename.

12.1.1.1 Example FORTRAN Call for Install ACP

```
return_status = CM50_ACP_INSTALL
                (%REF(acp_name),
                 %REF(process_name),
                 %REF(mailbox_name),
                 %REF(exe_path),
                 mode,
                 %REF(input_path),
                 %REF(output_path),
                 %REF(error_path),
                 %REF(privilege),
                 %REF(uic),
                 priority,
                 creprc_flags,
                 %REF(quota_list),
                 flags)
```

12.1.1.2 Parameter Definitions for Install ACP

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The CHARACTER*12 name of the ACP to be installed. Must be specified.

process_name—The CHARACTER*15 Name to be assigned to the created process. If set to spaces, the ACP name will be used. Note: Each process must have an unique name. The activation of an ACP will fail if a process with the specified process_name is active on the system.

mailbox_name—A CHARACTER*40 Mailbox name (normally set to spaces) to receive a termination message when the created process (ACP) is complete. This is a temporary termination mailbox created by the Programmatic Interface and ACPOPER utility. For more information, refer to the VMS System Services Reference Manual. This mailbox parameter is applicable only when the ACP is executed as a remote (detached) process. An ACP run interactively ignores the mailbox parameter in the ACP table. The mailbox is created using VMS defaults.

exe_path—CHARACTER*80 Full pathname of the executable file. If set to spaces, the default is the executable file specified by the acp_name in the CM50$ACP directory.

mode—BYTE value that specifies what mode to install the ACP in. The values are:

1 = **TEST**
2 = **RESTRICTED**
3 = **NORMAL**

input_path—CHARACTER*80 Pathname of the alternate input filename.

output_path—CHARACTER*80 Pathname of the alternate output filename. If left blank, SYS$OUTPUT will be directed to the NULL device.

error_path—CHARACTER*80 Pathname of the alternate error filename. If left blank, SYS$ERROR will be directed to the NULL device.

privilege—Privileges specification. Declared as RECORD /PRIV_MASK_TYPE/ assigns special VMS privileges to the ACP. Set both components (.L0 and .L1) to zero for a normal, unprivileged ACP.

uic—CHARACTER*12 name of user whose UIC is to be used when the ACP is executed remotely. Only the first 12 characters are significant; the remainder should be blank filled.

priority—INTEGER*4 specifying the VMS priority (0-30) of the ACP process.

creprc_flags—INTEGER*4 VMS Create Process flags specification. Normally set to zero.

quota_list—Quotas specification. Declare as an array of RECORD /QUOTA_TYPE/. The last element of the array must have a QUOTA_TAG = zero. To use the system defaults, pass a single element with a value of zero.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_ACIDP_ACTIVATE  (required to activate an ACP which is
                        connected to an ACIDP)
CM50$M_HANDLER
CM50$M_MSGON
```

12.1.2 Uninstall ACP

This routine is called to uninstall an ACP.

12.1.2.1 Example FORTRAN Call for Uninstall ACP

```
return_status = CM50_ACP_UNINST
                 (%REF(acp_name),
                  flags)
```

12.1.2.2 Parameter Definitions for Uninstall ACP

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The CHARACTER*12 name of the ACP that is to be uninstalled.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.1.3 Activate ACP

This routine is called to activate an installed ACP under a mode specified by the user.

12.1.3.1 Example FORTRAN Call for Activate ACP

```
return_status = CM50_ACP_ACT
                 (%REF(acp_name),
                  mode,
                  flags)
```

12.1.3.2 Parameter Definitions for Activate ACP

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The CHARACTER*12 name of the ACP to be activated.

mode—INTEGER*2 that specifies whether the ACP is to be activated as a REMOTE detached process (mode = 0) or as an INTERACTIVE subprocess (mode = 1).

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.1.4 Deactivate ACP

This routine is called to deactivate an installed ACP, placing it in a specified state.

12.1.4.1 Example FORTRAN Call for Deactivate ACP

```
return_status = CM50_ACP_DEACTIVATE
                 (%REF(acp_name),
                  state,
                  flags)
```

12.1.4.2 Parameter Definitions for Deactivate ACP

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The CHARACTER*12 name of the running ACP to be deactivated.

state—INTEGER*2 that specifies whether to set the ACIDP to an ABORT (state = 0) or OFF/DELAY (state = 3).

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.1.5 Connect ACP to an ACIDP

This routine is called to connect an installed ACP to an ACIDP on the LCN.

12.1.5.1 Example FORTRAN Call for Connect ACP to an ACIDP

```
return_status = CM50_ACP_CONNECT
                 (%REF(acp_name),
                  %REF(acidp_name),
                  cg_port_number,
                  flags)
```

12.1.5.2 Parameter Definitions for Connect ACP to an ACIDP

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The CHARACTER*12 name of the ACP to be connected.

acidp_name—The CHARACTER*16 name of the ACIDP to connect the to the ACP.

cg_port_number—INTEGER*2 that specifies which CG (1-4) contains the `ACIDP.`

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.1.6 Disconnect ACP from an ACIDP

This routine is called to disconnect an installed ACP from an ACIDP on the LCN.

12.1.6.1 Example FORTRAN Call for Disconnect ACP from an ACIDP

```
return_status = CM50_ACP_DISCON
                 (%REF(acp_name),
                  flags)
```

12.1.6.2 Parameter Definitions for Disconnect ACP from an ACIDP

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The CHARACTER*12 name of the ACP to be disconnected.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.1.7 Change ACP Mode

This routine is called to change the installation mode of an ACP.

12.1.7.1 Example FORTRAN Call for Change ACP Mode

```
return_status = CM50_ACP_CHG_MODE
                 (%REF(acp_name),
                  mode,
                  flags)
```

12.1.7.2 Parameter Definitions for Change ACP Mode

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The CHARACTER*12 name of the ACP whose mode is to be changed.

mode—INTEGER*2 that specifies the new mode of the ACP. Permitted values are:

1 = TEST
2 = RESTRICTED
3 = NORMAL

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

`CM50$M_HANDLER`
`CM50$M_MSGON`

12.1.8 Get ACP Summary

This routine is called to get summary information for an installed ACP. The output optionally can be sent to the printer.

12.1.8.1 Example FORTRAN Call for Get ACP Summary

```
return_status = CM50_ACP_SUM
                (%REF(acp_name),
                 %REF(acp_summary),
                 flags)
```

12.1.8.2 Parameter Definitions for Get ACP Summary

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The CHARACTER*12 name of the ACP for which summary information is to be returned.

acp_summary—This parameter specifies where the summary information is to be returned. Declared as a RECORD /ACP_SUMMARY_STRUCT/; the specific contents are described in the CM50_ACP_INCLUDE file.

Note that the ACP_SUMMARY_STRUCT record can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.1.9 Get List of ACPs

This routine is called to get a list of installed ACPs. Up to 400 ACPs will be reported in a single call, if more than 400 ACPs are installed on a system, repeating the call with a start_rec of 1, 401 and 801 will return additional ACPs until the system maximum of 1000 have been returned.

12.1.9.1 Example FORTRAN Call for Get List of ACPs

```
RETURN_STATUS = CM50_ACP_LISTALL
                (start_rec,
                 end_rec,
                 total_returned,
                 %REF(list),
                 flags)
```

12.1.9.2 Parameter Definitions for Get List of ACPs

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

start_rec—An INTEGER*4 specifying the starting (lowest) record number within the ACP status table to be reported.

end_rec—An INTEGER*4 specifying the ending (highest)record number within the ACP status table to be reported.

total_returned—An INTEGER*4 value specifying the number of records actually returned. This may be less than the number requested if the end of the table was reached.

list—An array of RECORD /ACP_SUMMARY_STRUCT/ receives the data requested. It must be dimensioned large enough to for the number of records requested (1 + end_rec - start_rec).

Note that the ACP_SUMMARY_STRUCT record can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.2 PROGRAMMATIC INTERFACE TO DDT OPERATIONS

All CM50S DDT operations except for `Edit` can be accessed through the Programmatic Interface. All Programmatic Interface routines are called as functions, and the status of each call is returned as the value of the function call. The calling program should include both CM50_FLAGS_INCLUDE.FOR and CM50_DDT_INCLUDE.FOR.

Exception handling is provided by standard VMS condition handling routines or by custom routines written by the user. The Programmatic calls for all DDT functions are described in detail in the following paragraphs.

12.2.1 Build/Rebuild DDT

This routine is called to build, or rebuild, a DDT binary file from a DDT source file. Flag options include CG residence for the DDT and DDT Rebuild.

12.2.1.1 Example FORTRAN Call for Build/Rebuild DDT

```
return_status = CM50_DDT_BUILD
                (%REF(ddt_name),
                 %REF(source_path),
                 cg_port_number,
                 %REF(description),
                 flags)
```

12.2.1.2 Parameter Definitions for Build/Rebuild DDT

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The CHARACTER*9 name of the DDT to be used to retrieve data. It may be left blank if the full source path is specified.

source_path—CHARACTER*80 pathname of the DDT source file. If set to spaces, the default is the DDT name in the current directory.

cg_port_number—INTEGER*2 which specifies which CG the DDT is associated with.

description—A CHARACTER*36 text description of the DDT being built. Note that if the DDT source file specifies a description, that description will be used and the value of this argument is ignored.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_REBUILD_DDT
CM50$M_DMP_DDT_ERRORS
CM50$M_MSGON
CM50$M_NO_SOURCE_DEBUG
CM50$M_CG_RES
CM50$M_WRITE_VT
```

NOTE

If the DDT (or another DDT by the same name) has already been built, then the CM50$M_REBUILD_DDT must be set ON.

For a new DDT, the CM50$M_REBUILD_DDT flag must be OFF.

12.2.2 Delete DDT

This routine is called to delete a DDT that already exists in the DDT table. If the DDT is installed in the CG, it is removed.

12.2.2.1 Example FORTRAN Call for Delete DDT

```
return_status = CM50_DDT_DELETE
                (%REF(ddt_name),
                 flags)
```

12.2.2.2 Parameter Definitions for Delete DDT

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The CHARACTER*9 name of the DDT to be deleted.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.2.3 Get DDT Summary

This routine is called to summarize the specifications of a particular DDT.

12.2.3.1 Example FORTRAN Call for Get DDT Summary

```
return_status = CM50_DDT_SUM
                (%REF(ddt_name),
                 %REF(summary),
                 flags)
```

12.2.3.2 Parameter Definitions for Get DDT Summary

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The CHARACTER*9 name of the DDT that is to be summarized.

summary—This argument (declared as RECORD /DDT_SUMMARY_STRUCT/) receives the requested information. Its contents are:

- Name of the DDT being summarized
- Pathname of the DDT's source file
- Description of the DDT
- Date that the DDT was first built
- Name of the original builder
- Most recent time the DDT was modified
- Installation status of the DDT
- Number of points in the DDT
- DDT Type—Input, Generic Input, Output, Generic Output, or History
- CG number that the DDT is associated with
- Whether or not DDT is installed in CG
- Name of ACIDP DDT is connected to
- Prefetch triggers

Note that the DDT_SUMMARY_STRUCT record can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.2.4 Get List of DDTs

This routine is called to retrieve a list of DDT summaries. Up to 500 DDT summaries can be returned in a single call. More than 500 DDTs can be retrieved by repeating this call with start_record set to 1, 501, 1001, etc., on successive calls.

12.2.4.1 Example FORTRAN Call for Get List of DDTs

```
return_status = CM50_DDT_LIST
                (start_record,
                 end_record,
                 count,
                 %REF(list),
                 flags)
```

12.2.4.2 Parameter Definitions for Get List of DDTs

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

start_record—This INTEGER*2 specifies the number of the first DDT to retrieve.

end_record—This INTEGER*2 specifies the number of the last DDT to retrieve.

count—This INTEGER*2 receives the actual number of DDT records returned to the caller.

list—This argument will receive an array of DDTs information. Declare as an array of up to 500 RECORD /DDT_SUMMARY_STRUCT/. The information returned for each record is

- Name of the DDT being summarized
- Pathname of the DDT's source file
- Description of the DDT
- Date that the DDT was first built
- Name of the original builder
- Most recent time the DDT was modified
- Installation status of the DDT
- Number of points in the DDT
- DDT Type—Input, Generic Input, Output, Generic Output, or History
- CG number that the DDT is associated with
- Tells whether DDT is installed in CG
- Name of ACIDP DDT is connected to

Note that the DDT_SUMMARY_STRUCT record can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The flags that apply to this call are `CM50$M_HANDLER` and `CM50$M_MSGON.`

12.2.5 Get DDT Detail

This routine is called to retrieve the detail information for the named DDT.

12.2.5.1 Example FORTRAN Call for Get DDT Detail

```
return_status = CM50_DDT_DETAIL
             (%REF(ddt_name),
              %REF(summary),
              %REF(data),
              %REF(points),
              %REF(details),
              %REF(values),
              flags)
```

12.2.5.2 Parameter Definitions for Get DDT Detail

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—CHARACTER*9 name of the DDT being summarized.

summary—A record declared as RECORD /DDT_SUMMARY_STRUCT/ containing:

`DDT_Name`	Name of the DDT being summarized
`DDT_Sourc_Loc`	Pathname to the DDT's source file
`DDT_Desc`	Description of the DDT
`Built_On`	Date that the DDT was first built
`Built_By`	Tells who the original builder was
`Modified_On`	Most recent time the DDT was modified
`DDT_Status`	Installation status of the DDT
`Number_of_Pts`	Number of points in the DDT
`DDT_Type`	Input, Output, or History DDT
`CG_Port_Num`	CG that the DDT is associated with
`In_CG`	Tells whether DDT is installed in CG

Note that the DDT_SUMMARY_STRUCT record can vary for different releases of CM50N, so programs using this call should be recompiled when CM50S is upgraded.

data—A record declared as RECORD /DATA_TYPE_STRUCT/ containing:

`DDT_Types`	Names data types found in the DDT
`TTL_Each_Type`	Counts for each data type found

points—An array of up to 300 RECORD /POINTS_STRUCT/ (one record for each point in the DDT) containing:

`Point_Name`	Point name
`Param_Name`	Parameter name (with index)

Note that the POINTS_STRUCT record can vary for different releases of CM50N, so programs using this call should be recompiled when CM50S is upgraded.

details—An array of up to 300 records (one per point) declared as RECORD /DETAIL_STRUCT/ containing:

`Process_Type`	Real, Integer, ASCII, Enumeration, or Ordinal
`Dest_Src`	Destination or Source offset value
`Test`	Use test Y/N and test data value
`BVS`	Bad value substitution Y/N and data
`Algo`	Algorithm number selection and data
`Limits`	Limit check Y/N and data

values—An array of up to 300 records (one per point) declared as RECORD /SUBST_STRUCT/. This argument will contain useful information only if full Table Processing (including a Values Table) is being used with the DDT. It contains the values from the last use of the DDT showing the values before and after table processing conversions. Any LCN Real data Bad Values are returned as zeros.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.2.6 Connect DDT to ACIDP

This routine is called to connect a DDT to an ACIDP for the purpose of enabling the Data Prefetch Function in the CG. The ACIDP-ACP connection must already exist and the DDT must be CG-resident and not already connected to an ACIDP.

The `ddt_name`, and either the `acp_name`, or `acidp_name` parameters are required in the call. The Schedule, PPS and Demand parameters also are required.

12.2.6.1 Example FORTRAN Call for Connect DDT to ACIDP

```
return_status = CM50_DDT_CONNECT
                 (%REF(ddt_name),
                  %REF(acidp_name),
                  %REF(acp_name),
                  %REF(trigger),
                  flags)
```

12.2.6.2 Parameter Definitions for Connect DDT to ACIDP

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The CHARACTER*9 name of the DDT that is to be connected to an ACIDP.

acidp_name—The CHARACTER*16 name of the ACIDP to which the DDT is to be connected. The `acidp_name` can be blanks if a valid `acp_name` is provided.

acp_name—The CHARACTER*12 name of the ACP connected to the ACIDP to which the DDT is to be connected. The `acp_name` can be blanks if a valid `acidp_name` is provided

trigger—CHARACTER*1 code with the three high-order bits assigned these meanings:

- Bit 7 : `Schedule`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 6 : `PPS` (Point_Process_Special)—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 5 : `Demand`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.2.7 Disconnect DDT from ACIDP

This routine is called to disconnect a DDT from an ACIDP. At least one of the three parameters, `ddt_name`, `acp_name`, or `acidp_name` is required in the call (the others are passed as blanks). The ACIDP-ACP-DDT connection must already exist.

12.2.7.1 Example FORTRAN Call for Disconnect DDT from ACIDP

```
return_status = CM50_DDT_DISCONNECT
                  (%REF(ddt_name),
                   %REF(acidp_name),
                   %REF(acp_name),
                   flags)
```

12.2.7.2 Parameter Definitions for Disconnect DDT from ACIDP

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The CHARACTER*9 name of the DDT that is to be disconnected. Can be blanks if either `acidp_name` or `acp_name` contains a valid name.

acidp_name—The CHARACTER*16 name of the ACIDP from which the DDT is to be disconnected. Can be blanks if either `ddt_name` or `acp_name` contains a valid name.

acp_name—The CHARACTER*12 name of the ACP connected to the ACIDP from which the DDT is to be disconnected. Can be blanks if either `ddt_name` or `acidp_name` contains a valid name.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.2.8 Modify Triggers

This routine is called to modify the Triggers associated with a DDT that is connected to an ACIDP. At least one of the three parameters, `ddt_name`, `acp_name`, or `acidp_name`, is required in the call (the others are passed as blanks). The ACIDP-ACP-DDT connection must already exist.

12.2.8.1 Example FORTRAN Call for Modify Triggers

```
return_status = CM50_DDT_TRIGGERS
                (%REF(ddt_name),
                 %REF(acidp_name),
                 %REF(acp_name),
                 %REF(trigger),
                 flags)
```

12.2.8.2 Parameter Definitions for Modify Triggers

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The CHARACTER*9 name of the DDT that is connected to the specified ACIDP. Can be blanks if either `acidp_name` or `acp_name` contains a valid name.

acidp_name—The CHARACTER*16 name of the ACIDP to which the specified DDT is connected. Can be blanks if either `ddt_name` or `acp_name` contains a valid name.

acp_name—The CHARACTER*12 name of the ACP connected to the specified ACIDP. Can be blanks if either `ddt_name` or `acidp_name` contains a valid name.

trigger—CHARACTER*1 code with the three high-order bits assigned these meanings:

- Bit 7 : `Schedule`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 6 : `PPS` (Point_Process_Special)—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 5 : `Demand`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.2.9 Install DDT Into CG

This routine is called to install the DDT into the CG.

12.2.9.1 Example FORTRAN Call for Install DDT Into CG

```
return_status = CM50_DDT_INSTALL
                 (%REF(ddt_name),
                  flags)
```

12.2.9.2 Parameter Definitions for Install DDT Into CG

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The CHARACTER*9 name of the DDT to be installed into the CG.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.2.10 Uninstall DDT from CG

This routine is called to remove a DDT from the CG.

12.2.10.1 Example FORTRAN Call for Uninstall DDT from CG

```
return_status = CM50_DDT_UNINST
                 (%REF(ddt_name),
                  flags)
```

12.2.10.2 Parameter Definitions for Uninstall DDT from CG

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The CHARACTER*9 name of the DDT to be removed from the CG.

flags—This INTEGER*4 parameter sets options as described in section 9.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

12.3 PROGRAMMATIC INTERFACE TO CG DATABASE

These functions return information about the current points configured in the database of any CG connected to the CM50. The language specific declarations for these functions are contained in `CM50_CGDATA_INCLUDE.for`

12.3.1 Resident DDT Summary

This function returns a list of all of the DDTs currently resident in the CG.

12.3.1.1 Example FORTRAN Function Call for Resident DDT List

```
return_status =  CM50_CG_RDDT
                 (cg_port_num,
                  number_of_values,
                  %REF(ddt_list))
```

12.3.1.2 Parameter Definitions for Resident DDT List

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of an INTEGER*2 identifying the CG to be accessed.

number_of_values—The name of an INTEGER*4 that returns the number of DDTs currently installed as resident in the CG.

ddt_list—The name of an array of up to 40 CHARACTER*10 variables that will contain the names of the resident DDTs.

12.3.2 Calculated Results Data Points List

This function returns a list of all of the CRDPs currently configured in the CG.

12.3.2.1 Example FORTRAN Call for CRDP List

```
return_status =  CM50_CG_CRDP
                 (cg_port_num,
                  number_of_values,
                  %REF(crdp_list))
```

12.3.2.2 Parameter Definitions for CRDP List

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of an INTEGER*2 identifying the CG to be accessed.

number_of_values—The name of an INTEGER*4 that returns the number of CRDPs currently configured in the CG.

crdp_list—The name of an array of 500 CHARACTER*8 variables that will contain the names of the CRDPs.

Note that the CRDP_LIST structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

12.3.3 ACIDP Detail

This function returns information about the current status of a specific ACIDP.

12.3.3.1 Example FORTRAN Call for ACIDP Detail

```
return_status =  CM50_CG_ADETAIL
                 (cg_port_num,
                  %REF(acidp_record))
```

12.3.3.2 Parameter Definitions for ACIDP Detail

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of an INTEGER*2 identifying the CG to be accessed.

number_of_values—The name of an INTEGER*4 that returns the number of DDTs currently installed as resident in the CG.

acidp_record—The name of a record declared as RECORD /CM50_ACIDP_REC/ with the following format:

ACIDP	: 8-character name of the ACIDP
DESC	: 24-character descriptor of the ACIDP
UNIT	: 2-character LCN Unit to which the ACIDP is assigned
KEYWORD	: 8-character LCN alias for the ACIDP
ACP	: 12-character name of the connected ACP
MODE	: 8-character enumerated value of the Program Mode
EXEC	: 8-character enumerated value of the Execution State
ACCES	: 8-character enumerated value of the Data Access Mode
DDT	: 9-character name of attached DDT
ACTYP	: 8-character enumerated value of the Activation Type
INHIB	: 8-character enumerated value of the Inhibit flag
STIME	: 8-character value of the Scheduled Start Time
PERIOD	: 8-character value of the Schedule Cycle Period
NXTTIM	: 18-character value of the Next Scheduled Activation Time
TAKEIP	: 4-character enumerated value of the Take_Initial_Path flag
RUNINIT	: 4-character enumerated value of the Run_on_Initialization flag
CONFWT	: 4-character enumerated value of the Confirm_Wait flag
CONFRQ	: 4-character enumerated value of the Confirm_Request flag
SCH	: 4-character enumerated value of the Schedule Activation flag
PPS	: 4-character enumerated value of Program_Special Activation flag
DMD	: 4-character enumerated value of Operator_Demand Activation flag
GROUP	: unsigned INTEGER*2 value of the Group code.

Note that the CM50_ACIDP_REC can vary for different releases of CM50N, so programs using this call should be recompiled when CM50S is upgraded.

12.3.4 ACIDP Summary

This function returns a list of all of the ACIDPs configured in the CG.

12.3.4.1 Example FORTRAN Call for ACIDP Summary

```
return_status =  CM50_CG_ACIDP
                 (cg_port_num,
                  number_of_values,
                  %REF(acidp_list),
                  %REF(acp_list),
                  mode_list,
                  state_list,
                  %REF(ddt_list),
                  trigger_list)
```

12.3.4.2 Parameter Definitions for ACIDP Summary

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of an INTEGER*2 identifying the CG to be accessed.

number_of_values—The name of an INTEGER*4 that returns the number of ACIDPs currently configured in the CG.

acidp_list—The name of an array of 250 CHARACTER*8 variables that will contain the names of the resident ACIDPs.

Note that the ACISP_LIST can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

acp_list—The name of an array of 250 CHARACTER*12 variables that will contain the names of the ACPs connected to the corresponding ACIDP.

mode_list—The name of an array of 250 INTEGER*2 variables that will contain the integer code for the installation mode of each ACIDP.

state_list—The name of an array of 250 INTEGER*2 variables that will contain the integer code for the current execution state of each ACIDP.

ddt_list—The name of an array of 250 CHARACTER*10 variables that will contain the names of the DDT (if any) connected to the corresponding ACIDP.

trigger_list—The name of an array of 250 Trigger records,where each Trigger record is an array of 3 boolean INTEGER*2 values (indicating by 1 or 0 whether or not the connected DDT will be prefetched when the ACIDP is triggered on Schedule, Operator_Demand, or PPS).

12.3.5 LCN Configuration

This function returns information about the LCN configuration parameters for a specified CG. Note that the specified CG must be running TDC 3000 release 400 or later for this function to get a successful return_status.

12.3.5.1 Example FORTRAN Call for LCN Configuration

```
return_status = CM50_CG_CONFIG
                (cg_prot_num,
                 %REF(CGconfig_record))
```

12.3.5.2 Parameter Definitions for LCN Configuration

return_status—The name of an Integer*4 to receive the overall return status of the function call. For fully successful calls, return_status = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_prot_number—The name of an INTEGER*2 identifying the CG (1 to 4) to be accessed.

cgconfig_record—The name of a record declared as RECORD /CM50_CGCONFIG_REC/ with the following fields:

LCN_VER	:	INTEGER*2 TDC 3000 software release level
LCN_REV	:	INTEGER*2 TDC 3000 revision level (future)
LCN_NODE	:	INTEGER*2 LCN node number of this CG
CG_VER	:	INTEGER*2 CG personality software release
CG_REV	:	INTEGER*2 CG personality software revision
TIME_SYNCH	:	INTEGER*2 CG Time synchronization period
CONFIRM_TIME	:	INTEGER*2 CG Time out for message confirm
CG_STATION	:	INTEGER*2 HDLC station number of the LCN
T1_TIME	:	INTEGER*2 CG T1 timer
N2_COUNT	:	INTEGER*2 CG Retry count
FLOAT_FORMAT	:	INTEGER*2 Floating point format (should be 2 for IEEE)
BAUD_RATE	:	INTEGER*2 enumeration (0= 1200; 1= 1760, 3= 2400, 5= 4800, 7= 9600, 8= 19200, 13= 38400, 14= 56700, 15= 76800)
TAG_SIZE	:	INTEGER*2 (0 for 8-character maximum, 1 for 16-characters)
HM_USER_MIN	:	INTEGER*2 number of minutes in a user average
HM_SHIFT_WK	:	INTEGER*2 number of shifts per week
HM_START_HR	:	INTEGER*2 daily/weekly averages starting hour (0 starts Sunday morning just after midnight)
HM_MONTH_TYP	:	INTEGER*2 (0 is calendar, 1 is 28-day cycles)
PINID	:	CHARACTER*2 identifier of this LCN for Network Gateway routing
DESCR	:	CHARACTER*40 CG descriptor on the LCN

12.4 PROGRAMMATIC INTERFACE TO FILE TRANSFER

These functions execute LCN file transfer commands programmatically. The calling program must include both the CM50_FLAGS_INCLUDE.FOR and the CM50_FTF_INCLUDE.FOR files in its source to insure that the functions and arguments are properly declared.

The DATAOUT facility allows the user, when requesting the execution of specific file transfer transactions, to place relevant data in the dataout or catalog file. This dataout file is a shared file by all concurrent users of file transfer. For example, user "Jones" requests a CM50_FILE_CATALOG transaction, the results of which are placed into the current dataout file. User "Smith" then requests a CM50_VOLUME_CATALOG transaction. These results also are placed into the same (current) dataout file.

CM50_FILE_CATALOG and CM50_VOLUME_CATALOG are the only file transfer operations that require a dataout file. Other file transfer transactions treat dataout as an option for journalizing activity.

12.4.1 Read LCN File

This procedure will transfer a single file from an LCN NET volume to CM50S. Wildcard transfers of files are not supported. This procedure also creates an "LCN ATTRIBUTES" file for every LCN file that is transferred. Multiple copies of the same file, within the same VMS directory, are not allowed. The version number of the attributes file should remain 1. For more information regarding file attributes refer to the WRITE file procedure.

12.4.1.1 Example FORTRAN Call for File Read

```
Return_status  =  CM50_LCN_READ
                     (%REF(lcn_file),
                      %REF(host_file),
                      %REF(acidp_name),
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.1.2 Parameter Definitions for File Read

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (CHARACTER*28) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx.

host_file—VMS pathname (CHARACTER*80) to be used to store the LCN file (and its associated attributes file). If no extension is specified, the VMS default of .DAT will be used. If no directory is specified, the user's current default directory will be used. The LCN attributes file uses the following naming convention: The filename suffix or extension is preceded by an under-bar character and followed by a period "LA" extension. For example; the LCN filename of FORMULAE.CL would have an attribute file of FORMULAE_CL.LA. Note: The transfer will fail if the pathname matches that of an existing file.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.2 Write LCN File

This procedure will transfer a single file from CM50S to LCN NET volume. An LCN ATTRIBUTES file is required for every LCN file that is transferred. Multiple copies of an LCN FILE within the same VMS directory are allowed. These files would have been created by modifying the original LCN FILE which was transferred as version 1. The version number of the attributes file should be 1.

12.4.2.1 Example FORTRAN Call for File Write

```
Return_status  =  CM50_LCN_WRITE
                     (%REF(host_file),
                      %REF(lcn_file),
                      %REF(acidp_name),
                      file_code,
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.2.2 Parameter Definitions for File Write

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

host_file—VMS pathname (CHARACTER*80) of the file to be transferred to the LCN If no directory is specified, the user's current default directory will be used. The associated LCN attributes file (with an extension of .LA) must be in the same directory.

lcn_file—LCN pathname (CHARACTER*28) where the file is to be stored on the LCN. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

file_code—Name of an INTEGER*2 variable that determines whether the LCN file is to be replaced if it already exists at the LCN NET volume. The default is to abort the write if the file already exists. The values are:

0: Replace existing file
1: Return an error if the file already exists.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.3 List LCN File Attributes

This request will retrieve the file attributes for a specific LCN file. Wildcard characters, and dataout options are not permitted. The file attributes are :

- Lcn file type—contiguous or linked
- Lcn file protection
- Record size
- Block size
- Lcn file configuration
- Lcn file revision
- Directory timestamp (Mo/Dd/Yr Mm:Ss)
- Logical number of blocks
- Logical number of records
- File Descriptor
- Starting Sector
- Ending Sector

12.4.3.1 Example FORTRAN Call for File Attributes

```
Return_status  =  CM50_ATTR_LIST
                     (%REF(lcn_file),
                      %REF(acidp_name),
                      %REF(attributes),
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.3.2 Parameter Definitions for File Attributes

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (CHARACTER*28) identifying the file whose attributes are to be returned. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

attributes—Buffer (declared as RECORD /FILE_ATTRIBUTE_BLOCK/, and described in CM50$LIB:CM50_FTF_INCLUDE.for) that will receive requested data.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.4 List LCN File Names

This transaction will retrieve up to 1180 file names and extensions from an LCN NET volume. If the number of files exceeds the buffer capacity of 1180, then multiple requests by directory, file type extension, or filename syntax must be used. Wildcards are permitted.

12.4.4.1 Example FORTRAN Call for List Files

```
Return_status  =  CM50_FILE_LIST
                     (%REF(lcn_file),
                      %REF(acidp_name),
                      %REF(file_list,
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.4.2 Parameter Definitions for List Files

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (CHARACTER*28) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats are:

NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

file_list—Buffer (declared as RECORD /FILE_LIST_ARRAY/, and described in CM50$LIB:CM50_FTF_INCLUDE.for) that will receive the list of file names and attributes.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.5 List LCN Volumes/Directories

This transaction will fetch from the History Module volume and directory names and sector usage figures. Wildcards and options are not permitted for this transaction type.

12.4.5.1 Example FORTRAN Call for List Volumes

```
Return_status  =  CM50_HM_LISTF
                     (%REF(cn_device),
                      %REF(acidp_name),
                      %REF(vol_record),
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.5.2 Parameter Definitions for List Volumes

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_device—LCN pathname (CHARACTER*28) identifying the device to be cataloged. Use the form PN:**nn** where **nn** is the lcn node number.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

vol_record—Buffer (declared as RECORD /VOLUME_STRUCTURES/, and described in CM50$LIB:CM50_FTF_INCLUDE.for) that will receive the Volume and directory information. This information includes:

- Number of Volumes
- Number of Sectors / Device
- Sectors in Use / Device
- Volume Name(s)
- Directory Name(s) on each volume

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.6 Cataloging LCN Files to Dataout

This file transfer transaction will list the LCN FILE ATTRIBUTES of one or more files into the current dataout file. The dataout file must have been previously established. The absence of a dataout specification will result in an error return.

Further processing requires that the dataout or catalog file be transferred to the VAX using the CM50_LCN_READ programmatic function.

12.4.6.1 Example FORTRAN Call for File Catalog

```
Return_status =  CM50_FILE_CATALOG
                     (%REF(lcn_file),
                      %REF(cat_file),
                      %REF(acidp_name),
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.6.2 Parameter Definitions for File Catalog

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (CHARACTER*28) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:

NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

Optional suffixes will increase the amount of information returned:

-FD will cause file descriptors to be listed
-REC will cause record and block data to be listed

cat_file—LCN pathname (CHARACTER*28) identifying the file to receive the catalog. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.7 Cataloging LCN Volumes to Dataout

This file transfer transaction will list the LCN Volumes and Directories for all History modules on the NET or for a specific History Module. The dataout file must have been previously established. The absence of a dataout specification will result in an error return.

12.4.7.1 Example FORTRAN Call for Volume Catalog

```
Return_status  =  CM50_VOLUME_CATALOG
                     (%REF(lcn_device),
                      %REF(cat_file),
                      %REF(acidp_name),
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.7.2 Parameter Definitions for Volume Catalog

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_device—LCN pathname (CHARACTER*28) identifying the device to be cataloged. Use the form NET or PN:**nn** where **nn** is the lcn node number.

cat_file—LCN pathname (CHARACTER*28) identifying the file to receive the catalog. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.8 LCN File Copy

This file transfer transaction will copy a single file or all files from one NET volume to another Net volume. The -D option is supported for journalizing all copies to a dataout file. The dataout file must have been previously established. Wildcards are permitted; however, the destination suffix must always be the same as the source suffix. Note that using the -D option without having previously defined a dataout path will result in an error and the copy function will not be completed.

12.4.8.1 Example FORTRAN Call for LCN File Copy

```
Return_status  =  CM50_LCN_COPY
                     (%REF(lcn_file),
                      %REF(out_file),
                      %REF(acidp_name),
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.8.2 Parameter Definitions for LCN File Copy

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (CHARACTER*28) identifying the file to be copied. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:

NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

out_file—Recipient filename (CHARACTER*28) specifying the pathname of the new file. The actions will be journalized if a DATAOUT file has been enabled and the "-D" option suffix is appended to the filename.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.9 LCN File Move

This file transfer transaction will move a single file or all files from one directory to another directory within the same NET volume. Wildcards are permitted and the -D option is supported for journalizing all moves to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the move function will not have been completed.

12.4.9.1 Example FORTRAN Call for LCN File Move

```
Return_status  =  CM50_LCN_MOVE
                     (%REF(lcn_file),
                      %REF(out_directory),
                      %REF(acidp_name),
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.9.2 Parameter Definitions for LCN File Move

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (CHARACTER*28) identifying the file to be moved. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:

NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

out_directory—Directory name (CHARACTER*28) specifying the directory to receive the moved file. This directory must be on the same HM volume as the original file. (The file name and extensions will remain unchanged.) The actions will be journalized if a DATAOUT file has been enabled and the " -D" option suffix is appended to the filename.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.10 LCN File Rename

This file transfer transaction will rename a single file or all files on the History Module. Wildcards are permitted and the -D option is supported for journalizing all renames to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the rename function will not have been completed.

12.4.10.1 Example FORTRAN Call for LCN File Rename

```
Return_status  =  CM50_LCN_RENAME
                     (%REF(lcn_file),
                      %REF(out_file),
                      %REF(acidp_name),
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.10.2 Parameter Definitions for LCN File Rename

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (CHARACTER*28) identifying the file to be renamed. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:

NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

out_file—Recipient filename (CHARACTER*28) specifying the new file name. (The directory and extensions will remain unchanged.) The actions will be journalized if a DATAOUT file has been enabled and the " -D" option suffix is appended to the filename.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.11 LCN File Delete

This file transfer transaction will delete a single file or all files from the specified volume on the History Module. Wildcards are permitted and the -D option is supported for journalizing all deleted files to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the delete file function will not be completed.
Once deleted the file cannot be recovered.

12.4.11.1 Example FORTRAN Call for LCN File Delete

```
Return_status  =  CM50_LCN_DELETE
                     (%REF(lcn_file),
                      %REF(acidp_name),
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.11.2 Parameter Definitions for LCN File Delete

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (CHARACTER*28) identifying the file to be copied. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:

NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

The actions will be journalized if a DATAOUT file has been enabled and the " -D" option suffix is appended to the pathname.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.12 LCN Directory Maintenance

These file transfer transactions will create or delete a directory under a volume on the History Module. No wildcards characters or options are permitted.

12.4.12.1 Example FORTRAN Call for Directory Maintenance

```
Return_status  = CM50_LCN_DIRECTORY
                          (%REF(lcn_directory),
                           action,
                           %REF(acidp_name),
                           cg_port_number,
                           lcn_sts,
                           flags)
```

12.4.12.2 Parameter Definitions for Directory Maintenance

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_directory—LCN pathname (CHARACTER*28) identifying the LCN directory to be created or deleted. Use the form NET>VDIR> DIR (Note the space delimiter before the directory name.)

action—An INTEGER*2 variable that specifies whether the named directory is to be created or deleted. The acceptable values are:

0 = `create_directory`
1 = `delete_directory`

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.13 Dataout Status

The dataout function allows the user, when requesting the execution of specific file transfer transactions, to place relative data in the dataout or catalog file. This dataout file is a shared file by all concurrent users of file transfer. For example, user "Jones" requests a CM50_FILE_CATALOG transaction, the results of which are placed into the current dataout file. User "Smith" then requests a CM50_VOLUME_CATALOG transaction. These results also are placed into the same (current) dataout file. The dataout file may be transferred to the VAX host using a CM50_LCN_READ request. The CM50_DATA_OUT transaction is provided to enable, disable, or query the file transfer dataout status.

12.4.13.1 Example FORTRAN Call for DATAOUT status

```
Return_status  =  CM50_DATA_OUT
                     (%REF(cat_file),
                      %REF(acidp_name),
                      do_action,
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.13.2 Parameter Definitions for DATAOUT status

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

cat_file—LCN pathname (CHARACTER*28) identifying the file to be used as the dataout journal. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

do_action—An INTEGER*2 variable that specifies the action to be taken. The values are:
0 = Disable dataout journaling
1 = Enable dataout journaling using the specified path
2 = Return the current dataout path

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

12.4.14 Abort File Transfer Transaction

This transaction CM50_ABORT_TRANSFER will terminate the current transaction in progress. The initiator of the transaction will receive a CM50_FTF_ABORT error return status. The initiator of the CM50_ABORT_TRANSFER request will receive a normal return status. No error is generated if there is not a current process to abort.

12.4.14.1 Example FORTRAN Call for Abort Transfer

```
Return_status  =  CM50_ABORT_TRANSFER
                     (%REF(acidp_name),
                      cg_port_number,
                      lcn_sts,
                      flags)
```

12.4.14.2 Parameter Definitions for Abort Transfer

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

acidp_name—A CHARACTER*16 ACIDP name reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of an INTEGER*2 variable that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of an INTEGER*2 variable which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—INTEGER*4 parameter (optionally declared as CM50_FLAG_TYPE) that sets options as described in section 9.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

PASCAL LANGUAGE CONSIDERATIONS
Section 13

This section discusses each of the program interfaces that provide necessary services that enable Pascal programs to communicate with other nodes on the TDC 3000 Local Control Network.

13.1 CM50S INCLUDE FILES

Each user interface routine has language-specific interfaces that are supported by include files that contain data declarations that match the argument names and symbolic constants used in the example calls in this section. **Any program that uses any of these interface routines should be compiled with the matching language-specific include files.**

13.1.1 Data Transfer Functions

These include files generally are needed by ACPs and Data Acquisition programs.

`CM50$LIB:CM50_INCLUDE.PAS`	Contains the declarations used by the LCN data interfaces (Sections 14 & 15) and the Vintage Routines (Appendix G).
`CM50$LIB:CM50_ERROR_INCLUDE.PAS`	Contains the symbolic names for all of the CM50S error codes (Appendix A.2).
`CM50$LIB:CM50_FTF_INCLUDE.PAS`	Definitions for all LCN file transfer operations (section 16.4)

13.1.2 DDT and ACP Management

These include files are needed by applications that use the CM50S administration, calls described in Section 16. DDT and ACP management functions use some shared data structures, that are defined in the file CM50_FLAGS_INCLUDE.PAS. Therefore, that file should be included with any any program that calls either DDT or ACP functions and must precede the include file defining those specific functions.

`CM50$LIB:CM50_FLAGS_INCLUDE.PAS`	Definitions for the shared data structures in the ACP & DDT Management Interface calls. Must be included prior to either CM50_ACP_INCLUDE.PAS or CM50_DDT_INCLUDE.PAS.
`CM50$LIB:CM50_ACP_INCLUDE.PAS`	Definitions for all ACP Management operations (section 16.1).
`CM50$LIB:CM50_DDT_INCLUDE.PAS`	Definitions for all the DDT Management operations (section 16.2).
`CM50$LIB:CM50_CGDATA_INCLUDE.PAS`	Definitions for all the CG Database retrievals (section 16.3).

13.1.3 Programmatic Interface Flag Parameters

An INTEGER*4 parameter called FLAGS is included in every ACP and DDT management function to control some of the handling options. Some of the flags apply to only the DDT calls, some to only the ACP calls, and some can be used by both. All user-visible flags (as defined in CM50_FLAGS_INCLUDE.PAS) are described below.

- CM50$M_HANDLER—(Bit 0) Indicates that the user has provided a custom exception handler. The default is OFF.
- CM50$M_MSGON—(Bit 1) Prints all diagnostic messages to SYS$OUTPUT. The default is OFF.
- CM50$M_CGRES—(Bit 5) Installs the DDT as CG resident. The default is OFF.
- CM50$M_REBUILD_DDT—(Bit 6) Rebuilds an existing DDT. The default is OFF.
- CM50$M_NO_SOURC_DEBUG—(Bit 7) Produces no error file during DDT build. The default is OFF.
- CM50$M_DMP_DDT_ERRORS—(Bit 8) Sends the error file produced by a DDT build to SYS$OUTPUT after building the DDT. If set, then the CM50$M_NO_SOURCE_DEBUG flag must be OFF.
- CM50$M_ACIDP_ACTIVATE—(Bit 9) Reserved for internal CM50S use.
- CM50$M_WRITE_VT—(Bit 10) Creates the .VT file with write privilege.

All of the flags described above, represent bit masks that can be added together to enable any combination of the flags. These flag values also can be used to see if a particular flag is set. An example is shown below.

```
     flags := CM50$M_HANDLER + CM50$M_MSGON;
    Return_Status = DDT_SUMMARY(DDT_Name,
-                               Summary, Flags)
```

13.2 CALLING CONVENTIONS

CM50S interface routines follow the VMS language-independent calling conventions. With the exception of some housekeeping procedures that have no error handling (such as ACPTRP and PRGTRM), they are written as functions.

We recommend that each function call be followed by a logical test of the return_status value. If ODD(return_status) is true, the call was successful (although individual data items may require checking); otherwise (even valued status codes), appropriate error handling should be invoked. Note that if the application does not check return_status, the interface routine can be invoked as a called subroutine or procedure in the same manner as VMS system services.

All the calling sequence examples shown in this section are appropriate for Pascal programs.

A VMS shortword is a 16-bit integer (declared as CM50$int2).

Arguments should be declared 'var' of the appropriate type, so that their addresses (pass by REFerence) are passed instead of their values.

Arguments should normally be declared using the types shown in the documentation. These declarations support the maximum array sizes that CM50S will process. Compatible types (different sizes of arrays, for examples) may be used provided that the argument is passed with the explicit %REF qualifier.

Variant arguments (e.g., enumeration/ordinal arrays) are normally passed using the explicit %REF qualifier (to bypass internal testing of data typing).

Boolean (True/False) arguments are shortwords (declared as CM50$bool2), with a value of 1 for True and 0 for False.

13.3 COMPATIBILITY OF APPLICATION PROGRAM WITH ITS DDTS

Because each application program and its Data Definition Tables (and Multi_Point List structures) are separately built, the system cannot enforce compatibility between a program and any DDT(s) that it uses. That responsibility is up to you.

In particular, it is vital that the dimensions set for data-receiving arrays be large enough to accommodate the maximum data amounts permitted by the named DDT.

Specific points to remember for DDT Get Data and DDT Store Data are

- Dimensions set for each value-type's program array must be equal-to or greater-than the value-type's point count in the referenced DDT. The values can be stored one-for-one or they can be scattered as defined in the DDT. If the program arrays are too small, data or program code may be corrupted (DDT Get Data) or inappropriate data may be exported (DDT Store Data).

- The dimension values for status table arrays must be equal-to or greater-than the total number of points of all types in the referenced DDT because this array is to receive a status code for each returned value, positioned according to its location in the DDT.

13.4 DATA REPRESENTATIONS

Differences between data representations in the VAX and the CG normally are resolved by the CG-VAX Communications Handlers, thus are invisible to the user (Exception: raw data transfers, see heading 14.3). The LCN data formats are:

Real—32 bit floating point matches normal REAL format except that bad values (NaN) from the LCN have the bit pattern for -0. This value will cause a Pascal trap if used in an arithmetic or assignment statement, so real values returned from the LCN should always be tested (using either the CM50_VALIDN function or the associated value_status_table entry for the value).

Integer—shortword value -32768..+32767.

ASCII—packed array of 24 characters.

String—packed array of 40 characters.

Enumeration—There are two ways to represent LCN enumerations: as a packed array of 8 ASCII characters (Enumerated) or as shortword integer values (Ordinal). The choice of representation is made when the data transfer is requested, except that **self-defined enumerations should be transferred only as Ordinals**. For information on standard enumerations, see the *Application Module Parameter Reference Dictionary, Hiway Gateway Parameter Reference Dictionary*, and *Computer Gateway Parameter Reference Dictionary*. For information on Custom Data Segments, see the *System Control Functions* manual. For information on self-defined enumerations, see Section 2 of the *Hiway Gateway Control Functions* manual.

Time—LCN Internal Time is defined as a record structure (CM50_Time_Vals) consisting of an integer count of Seconds (since the start of 1979) followed by a shortword count of Ticks (tenths of milliseconds). Some of the calls will return LCN External Time, an ASCII string of format MM/DD/YYΔHH:MM:SS, where Δ represents a space. See heading 15.3.3 for time format conversions.

Entity ID—Internally stored as an 64-bit value (CM50_PTID_VALS type) identifying a specific point (Ptid or Internal_id). Also can be retrieved as a packed array of 18 characters (External_id) consisting of the up to 16-character point name followed by the two-character pinid for Network Gateway references.

13.5 COMMONLY MADE ERRORS

- Arguments must be declared as the correct data type. If string constants are used for arguments, they must be padded with spaces (or terminated with a null character). Use of the wrong length for a string will probably result in a Pascal runtime error. Make sure the %REF qualifier is used everywhere it is shown in the examples.

- Failure to use the 'VAR' specification in declaring a CM50S function. This will be avoided in the appropriate Include files are used.

- Failure to use the CM50_SET_ACP function (or ACPTRP call) as the first executable program statement of an ACP and/or failure to use the PRGTRM call as the last executable statement of an ACP.

- Attempting to run an application program with unresolved compile or link errors or the use of a DDT that is incomplete or complete with errors.

- Failure to specify array sizes and data types that match DDT definitions.

- Failure to specify all parameters required by the interface routines.

- Attempting to activate an ACP through an ACIDP while the ACP is linked to the VMS DEBUG utility. Use of the DEBUG utility is supported only for execution of ACPs while run interactively from a terminal.

- Terminating an ACP by use of the STOP/IDENTIFIER function of VMS DCL. ACPs should only be aborted through the CM50 Deactivate ACP procedure.

13.6 ERROR DETECTION BY INTERFACE FUNCTIONS

There are three categories of error that can be detected during the execution of a program when using the interface functions. These are indicated through one of these methods:

- Request completion status code

- Individual parameter status codes

- Program abort

The RETURN_STATUS value returned by the Function shows whether or not the request was successfully processed and, if not, what error type was involved. Some typical errors flagged by the return status are

- LCN access problems or data link failure

- ACP installation or mode problems

- Data problems in the call or with a referenced DDT

- Call rules violations

The RETURN_STATUS code follows the standard VAX/VMS condition status code format. In general, even number codes indicate fatal system problems or program bugs, while odd number codes indicate success (code 000000001) or partial success (e.g., code 215000051). See Appendix A.2 for additional information and a listing of all RETURN_STATUS values and their meanings.

Most of the interface calls also return LCN point.parameter values that are to be processed by the calling application program. Accompanying each value (or array) is a status code that must be checked for indications of problems that would invalidate the requested data. See the call arguments STATUS_TABLE or VALUE_STATUS in the individual interface descriptions. There are over 200 different data access-status codes that can be returned. See Appendix A.1 for a listing of these codes.

Some errors in use of the interface routines result in the application program being aborted. An error message is logged at the VAX operator console and is shown on the Universal Station Detail Display for a connected ACIDP. These errors can be of the following types:

- File access errors
- Communication Interface errors
- Format conversion errors
- Various program logic errors

13.7 SUMMARY OF USER-PROGRAM INTERFACES

Heading	Interface Descriptions	Function Names
	Multipoint (DDT) Data Transfers	
14.1.1	DDT Get Data	CM50_DDT_GET CM50_DDT_GETNT
14.1.2	DDT Store Data	CM50_DDT_STORE CM50_DDT_STORENT
14.1.3	Generic DDT Get Data	CM50_DDT_GETGEN
14.1.4	Generic DDT Store Data	CM50_DDT_STOREGEN
14.1.5	Multi-Point List Get Data	CM50_MPL_GET
14.1.6	Multi-Point List Store Data	CM50_MPL_STORE
14.1.7	Generate Multi-Point List	CM50_MPL_GENTLIST CM50_MPL_GENTAGS CM50_MPL_GENFILE
14.1.8	Read Multi-Point List	CM50_MPL_READ
14.1.9	Write Multi-Point List	CM50_MPL_WRITE
14.1.10	Create Include File for Multi-Point List	CM50_MPL_GENINCL
	Point List Data Transfers	
14.2.1	Point List Get Values	CM50_GET_PT_LIST
14.2.2	Point List Get by Value Type	
	Real Values	CM50_GET_REALNBR
	Integer Values	CM50_GET_INTNBR
	ASCII Values	CM50_GET_ASC24
	Enumeration Values	CM50_GET_ENUM
	Ordinal Values	CM50_GET_ORD
	Internal IDs	CM50_GET_PTID
	External IDs	CM50_GET_EXID
	Time Values	CM50_GET_TIME
	String Values	CM50_GET_STRI
14.2.3	Point List Store Values	CM50_STORE_PT_LIST
14.2.4	Point List Store by Value Type	
	Real Values	CM50_STORE_REALNBR
	Integer Values	CM50_STORE_INTNBR
	ASCII Values	CM50_STORE_ASC24
	Enumeration Values	CM50_STORE_ENUM
	Ordinal Values	CM50_STORE_ORD
	Internal Ids	CM50_STORE_PTID
	Time Values	CM50_STORE_TIME
	String Values	CM50_STORE_STRI
	Single Point Data Transfers	
14.3.1	Single Point Get Data(External ID)	CM50_GET_ID CM50_GET_TAG
14.3.2	Single Point Store Data(External ID)	CM50_STORE_ID CM50_STORE_TAG
14.3.3	Single Point Get Data (Internal ID)	CM50_GETPT_ID
14.3.4	Single Point Store Data (Internal ID)	CM50_STOREPT_ID
14.3.5	Get LCN Clock Value	CM50_TIMNOW_LCN CM50_TIMNOW_ASC
	Raw Data Transfers	
14.4.1	Raw Data Get	CM50_SPGRAW
14.4.2	Raw Data Store	CM50_SPSRAW
14.4.3	Convert Raw Data	CM50_SPCRAW

Heading	Interface Descriptions	Function Names
	History Data Transfers	
14.5.2	Get History Snapshots (Relative Time)	CM50_DDTHIS_SNAP CM50_DDTHIS_FAST CM50_MPLHIS_SNAP CM50_PTHIS_SNAP
14.5.3	Get History Snapshots (Absolute Time)	CM50_DDTHIS_SNAPT CM50_DDTHIS_FASTT CM50_MPLHIS_SNAPT CM50_PTHIS_SNAPT
14.5.4	Get History Averages (Relative Time)	CM50_DDTHIS_AVER CM50_MPLHIS_AVER CM50_PTHIS_AVER
14.5.5	Get History Averages (Absolute Time)	CM50_DDTHIS_AVERT CM50_MPLHIS_AVERT CM50_PTHIS_AVERT
14.5.6	Get Monthly Averages (Relative Time)	CM50_DDTHIS_MNTH CM50_MPLHIS_MNTH CM50_PTHIS_MNTH
14.5.7	Get Monthly Averages (Absolute Time)	CM50_DDTHIS_MNTHT CM50_MPLHIS_MNTHT CM50_PTHIS_MNTHT
14.5.8	Find History Collection Rate	CM50_DDTHIS_RATE CM50_MPLHIS_RATE CM50_PTHIS_RATE CM50_TAGHIS_RATE
	Text Message Transfers	
14.6.1	Get Message	CM50_GETMSG
14.6.2	Send Message	CM50_STOREMSG
	ACP Execution Support	
15.1.1	ACP Initialization	CM50_SET_ACP
15.1.2	Get ACP Status	GETSTS*
15.1.3	ACP Delay	CM50_ACPDELAY
15.1.4	ACP Hibernate	CM50_HIBER
15.1.5	ACP Termination	PRGTRM*
	Entity Name Conversions	
15.2.1	Convert External to Internal ID	CM50_CONV_PT CM50_CONV_TAG
15.2.2	Convert List of External IDs	CM50_CONV_PT_LIST CM50_CONV_TAG_LIST
	Value Conversions	
15.3.1	Valid Number Check	CM50_VALIDN
15.3.2	Set Bad Value	CM50_SETBAD
15.3.3	Convert Time Values	CM50_TIMLCN_ARY CM50_TIMLCN_ASC CM50_TIMLCN_EURO CM50_TIMLCN_VAXA CM50_TIMLCN_VAXB CM50_TIMARY_LCN CM50_TIMARY_ASC CM50_TIMARY_EURO CM50_TIMARY_VAXA CM50_TIMARY_VAXB CM50_TIMASC_LCN

* GETSTS and PRGTRM do not have a RETURN_STATUS, so they cannot be used as functions, but must be invoked as procedures.

Heading	Interface Descriptions	Function Names
15.3.3	Convert Time Values—continued	CM50_TIMASC_ARY CM50_TIMASC_EURO CM50_TIMASC_VAXA CM50_TIMASC_VAXB CM50_TIMEURO_LCN CM50_TIMEURO_ARY CM50_TIMEURO_ASC CM50_TIMEURO_VAXA CM50_TIMEURO_VAXB CM50_TIMVAXA_LCN CM50_TIMVAXA_ARY CM50_TIMVAXA_ASC CM50_TIMVAXA_EURO CM50_TIMVAXA_VAXB CM50_TIMVAXB_LCN CM50_TIMVAXB_ARY CM50_TIMVAXB_ASC CM50_TIMVAXB_EURO CM50_TIMVAXB_VAXA
	ACP Management	
16.1.1	Install an ACP	CM50_ACP_INSTALL
16.1.2	Uninstall an ACP	CM50_ACP_UNINST
16.1.3	Activate (run) an ACP	CM50_ACP_ACT
16.1.4	Deactivate (abort) an ACP	CM50_ACP_DEACTIVATE
16.1.5	Connect an ACP to an ACIDP	CM50_ACP_CONNECT
16.1.6	Disconnect ACP from its ACIDP	CM50_ACP_DISCON
16.1.7	Change ACP installation mode	CM50_ACP_CHG_MODE
16.1.8	Get ACP summary	CM50_ACP_SUM
16.1.9	Get list of ACPs	CM50_ACP_LISTALL
	DDT Management	
16.2.1	Build/Rebuild a DDT	CM50_DDT_BUILD
16.2.2	Delete a DDT	CM50_DDT_DELETE
16.2.3	Get DDT summary information	CM50_DDT_SUM
16.2.4	Get list of DDT summaries	CM50_DDT_LIST
16.2.5	Get DDT detailed information	CM50_DDT_DETAIL
16.2.6	Connect a DDT to an ACIDP	CM50_DDT_CONNECT
16.2.7	Disconnect a DDT from its ACIDP	CM50_DDT_DISCONNECT
16.2.8	Modify DDT prefetch triggers	CM50_DDT_TRIGGERS
16.2.9	Install a DDT as CG resident	CM50_DDT_INSTALL
16.2.10	Remove a DDT from CG residency	CM50_DDT_UNINST
	CG Database Routines	
16.3.1	Get list of resident DDTs	CM50_CG_RDDT
16.3.2	Get list of CRDPs	CM50_CG_CRDP
16.3.3	Get detailed ACIDP information	CM50_CG_ADETAIL
16.3.4	Get list of ACIDPs	CM50_CG_ACIDP
16.3.5	Get LCN Configuration	CM50_CG_CONFIG
	File Transfer Routines	
16.4.1	Read File from LCN	CM50_LCN_READ
16.4.2	Write File to LCN	CM50_LCN_WRITE
16.4.3	List LCN File Attributes	CM50_ATTR_LIST
16.4.4	List LCN Files & Extensions	CM50_FILE_LIST
16.4.5	List LCN Volumes/Directories	CM50_HM_LIST

16.4.6	List LCN Files to Dataout	CM50_FILE_CATALOG
16.4.7	List LCN Volumes to Dataout	CM50_VOLUME_CATALOG
16.4.8	LCN File Copy	CM50_LCN_COPY
16.4.9	LCN File Move	CM50_LCN_MOVE
16.4.10	LCN File Rename	CM50_LCN_RENAME
16.4.11	LCN File Delete	CM50_LCN_DELETE
16.4.12	LCN Directory Maintenance	CM50_LCN_DIRECTORY
16.4.13	LCN Dataout Status	CM50_DATA_OUT
16.4.14	Abort LCN File Transfer	CM50_ABORT_TRANSFER

LCN DATA TRANSFERS (Pascal)
Section 14

This section discusses program interfaces for Pascal programs to transfer data between the host VAX computer and the TDC 3000 Local Control Network.

14.1 MULTIPOINT (DDT) DATA TRANSFERS

The interface routines in this group require the use of separately prepared Data Definition Tables (DDT) that specify which points are to be accessed and what pre/post processing is to be done on data values. See Section 6 for DDT preparation and installation details.

Each DDT may reference a maximum of four different data types. The standard DDT functions assume the data types are grouped into a "normal" order. It is possible to build DDTs with unusual combinations of data types that do not follow these assumptions. These special-case DDTs are tagged as GenIn (Generic Input) or GenOut (Generic Output) and may only be used with the Generic DDT Transfers described in Sections 14.1.3 and 14.1.4. Standard Input and Output DDTs may be used with either the Generic DDT transfers or the traditional DDT data interface routines.

Single elements of parameter arrays (but not whole arrays) can be specified in the DDT.

14.1.1 DDT Get Data Interface

This routine fetches data from the DDT's associated CG or elsewhere on its LCN. The specification of which data is to be fetched and where it is to be stored in the calling program's data arrays is contained in the Data Definition Table referenced by the call.

14.1.1.1 Example Pascal Call for DDT Get Data

```
return_status := CM50_DDT_GET  or CM50_DDT_GETNT
                 ddt_name,
                  real_values_array,
                  intg_values_array,
                   or  %REF ptid_values_array,
                   or  %REF time_values_array,
                  asci_values_array,
                   or  %REF string_values_array,
                   or  %REF exid_values_array,
                  %REF enum_array,
                   or  %REF ord_array,
                  status_table);
```

Use the Interface Name `CM50_DDT_GET` if you want data transformation operations performed by the Table Processor, and `CM50_DDT_GETNT` if you do not want data transformation operations performed (to decrease processing time).

The DDT Get Data call must specify four data types in the order shown (three of these can be dummy arguments that receive no data). Note that there are restrictions on the combinations of data types.

14.1.1.2 Parameter Definitions for DDT Get Data

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (declared as type cm50$DDT_STRING) that contains the name of the input Data Definition Table to be used.

real_values_array—The name of a Real array (declared as type cm50$real300) where the fetched Real values are to be stored. Bad values are returned as NaN (-0).

intg_values_array—The name of a shortword array (declared as type cm50$int300) where the fetched Integer values are to be stored.

ptid_values_array—The name of an array of internal entity ids (declared as cm50_Ptid_array_type).

time_values_array—The name of an array of LCN internal time values (declared as cm50_Time_array_type).

asci_values_array—The name of an array of 24-character strings (declared as cm50_asci_array_type) where the fetched ASCII values are to be stored. Bad values are returned as strings of question marks.

string_values_array—The name of an array of 40-character strings (declared as cm50_Stri_array_type) where the fetched LCN string values are to be stored.

exid_values_array—The name of an array of 18-character strings (declared as cm50_Exid_array_type) where the fetched external entity names are to be stored.

enum_array—The name of an array of 8-character strings (declared as type cm50_AEnm_array_type) where the fetched Enumeration values are to be stored. Bad values are returned as strings of question marks.

ord_array—The name of a shortword array (declared as type cm50$INT300) where the fetched ordinal values of enumerations are to be stored.

status_table—The name of a shortword array (declared as type cm50$INT300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A.1 for a listing of Data Access error/status codes.

14.1.2 DDT Store Data Interface

This routine sends data to points in the DDT's associated CG or elsewhere on its LCN. The specification of what points are to receive data and the location of data within the calling program's data arrays is contained in the Data Definition Table referenced by the call. Errors encountered during execution of the routine as well as individual point-data errors are returned to the calling program.

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

14.1.2.1 Example Pascal Call for DDT Store Data

```
return_status := CM50_DDT_STORE  or  CM50_DDT_STORENT
                 (ddt_name,
                  real_values_array,
                  intg_values_array,
                   or  %REF ptid_values_array,
                   or  %REF time_values_array,
                  asci_values_array,
                   or  %REF string_values_array,
                  %REF enum_array,
                   or  %REF ord_array,
                  store_array,
                  status_table)
```

Use the Interface Name CM50_DDT_STORE if you want data transformation operations performed by the Table Processor and CM50_DDT_STORENT if you do not want transformation operations performed (to decrease processing time).

The DDT Store Data call must specify four data types in the order shown (three of these can be dummy arguments that export no data). Note that there are restrictions on the combinations of data types.

14.1.2.2 Parameter Definitions for DDT Store Data

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `status_table` status code for each requested store value must be checked.

ddt_name—The name of a 9-character string (declared as type cm50$DDT_STRING) that contains the name of the output Data Definition Table to be used in the "Store Data" operation.

real_values_array—The name of a Real array (declared as type cm50$real300) that contains the Real values to be stored.

intg_values_array—The name of a shortword array (declared as type cm50$int300) that contains the Integer values to be stored.

ptid_values_array—The name of an array of internal entity ids (declared as cm50_Ptid_array_type) .

time_values_array—The name of an array of LCN internal time values (declared as cm50_Time_array_type) .

asci_values_array—The name of an array of 24-character strings (declared as cm50_Asci_array_type) that contains the ASCII values to be stored.

string_values_array—The name of an array of 40-character strings (declared as cm50_Stri_array_type) where LCN string values are stored.

enum_array—The name of an array of 8-character strings (declared as type cm50$int1200) that contains the Enumeration values to be stored. Use of enumeration strings by Store Data is limited to standard enumerations (including Custom Data Segments). All self-defined enumerations (such as digitals) must be accessed through their ordinal values.

ord_array—The name of a shortword array (declared as type cm50$int300) that contains the Ordinal values of Enumerations to be stored.

store_array—The name of a shortword array (declared as type cm50$int300) that contains a control code entry for each value to be stored. These codes control what—if any—value is to be stored. The store code values are

0 – Store the value from the Values Array
1 – Store the bad value representation instead
2 – Do not store any value.

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

status_table—The name of a shortword array (declared as type cm50$int300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A for a listing of Data Access error/status codes.

14.1.3 Generic DDT Get Data Interface

This routine fetches data for any Input or Generic Input DDT. The specification of which data is to be fetched and where it is to be stored in the calling program's data arrays is contained in the Data Definition Table referenced by the call.

14.1.3.1 Example Pascal Call for Generic DDT Get

```
return_status := CM50_DDT_GETGEN
                 (ddt_name,
                  values_array1,
                  values_array2,
                  values_array3,
                  values_array4,
                  status_table,
                  tbl_proc);
```

14.1.3.2 Parameter Definitions for Generic DDT Get

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (declared as type cm50$DDT_STRING) that contains the name of the Data Definition Table to be used.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array where the fetched values are to be stored. The data type for each array must match the corresponding data type in the DDT definition. Each array should be declared as a CM50$VALUE_RECORD type.

status_table—The name of a shortword array (declared as type cm50$int300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A.1 for a listing of Data Access error/status codes.

tbl_proc—The name of a Boolean shortword (declared as type cm50$int2) that determines whether or not table processing is to be suppressed. If tbl_proc is set to 1, all table processing (saving values to disk and/or data transformations) will be suppressed. Use a value of 0 for normal processing.

14.1.4 Generic DDT Store Data Interface

This routine sends data to points defined in any Output or Generic Output DDT. The specification of what points are to receive data and the location of data within the calling program's data arrays is contained in the Data Definition Table referenced by the call. Errors encountered during execution of the routine as well as individual point-data errors are returned to the calling program.

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

14.1.4.1 Example Pascal Call for Generic DDT Store

```
return_status := CM50_DDT_STOREGEN
                 (ddt_name,
                  values_array1,
                  values_array2,
                  values_array3,
                  values_array4,
                  store_array,
                  status_table,
                  tbl_proc);
```

14.1.4.2 Parameter Definitions for Generic DDT Store

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `status_table` status code for each requested store value must be checked.

ddt_name—The name of a 9-character string (declared as type cm50$DDT_STRING) that contains the name of the Data Definition Table to be used in the "Store Data" operation.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array that contains the data to be stored. The data type for each array must match the corresponding data type in the DDT definition. Each array should be declared as a CM50$VALUE_RECORD type.

store_array—The name of a shortword array (declared as type cm50$int300) that contains a control code entry for each value to be stored. These codes control what—if any—value is to be stored. The store code values are

- 0 – Store the value from the Values Array
- 1 – Store the bad value representation instead
- 2 – Do not store any value.

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

status_table—The name of a shortword array (declared as type cm50$int300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A for a listing of Data Access error/status codes.

tbl_proc—The name of a shortword (declared as type cm50$int2) that determines whether or not table processing is to be suppressed. If tbl_proc is set to 1, all table processing (saving values to disk and/or data transformations) will be suppressed. Use a value of 0 for normal processing.

14.1.5 Multi-Point List Get Data Interface

This routine fetches data for the LCN tags specified in an internal data block. An internal Data Block is a memory-resident equivalent of a DDT. The specification of which data is to be fetched and where it is to be stored in the calling program's data arrays can be prepared using any of the generate MPL routines (see 14.1.7) or you can read in a DDT from its disk file (see 14.1.8).

14.1.5.1 Example Pascal Call for Multi-Point List Get

```
return_status := CM50_MPL_GET
                 (mpl_name,
                  acidp_name,
                  %REF values_array1,
                  %REF values_array2,
                  %REF values_array3,
                  %REF values_array4,
                  status_table,
                  cg_port_num);
```

14.1.5.2 Parameter Definitions for Multi-Point List Get

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (CM50_ACP_RUN)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (CM50_LCN_PART)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

mpl_name—The name of a Multi-Point List structure (declared as type cm50_idb_rec) defining the data to be retrieved.

acidp_name—A 16-character string (declared as type cm50_long_ACIDP) containing the name of an ACIDP. If the ACIDP is spaces, then the data will be retrieved without any ACIDP controls. If an ACIDP is named, then the data access will be completed only if that ACIDP is in RUN state.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array where the fetched values are to be stored. The data type for each array must match the corresponding data type in the MPL definition. Each array should be declared as a CM50$VALUE_RECORD type.

status_table—The name of a shortword array (declared as type cm50$int300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag in the list. See Appendix A.1 for a listing of Data Access error/status codes.

cg_port_num—The name of a shortword (declared as type cm50$int2 with a value of 1-4) identifying the CG to be accessed.

14.1.6 Multi-Point List Store Data Interface

This routine stores data for the LCN tags specified in an internal data block. An internal Data Block is a memory-resident equivalent of a DDT. The specification of which tags are to receive data and the location of the values within the calling program's data arrays can be prepared using any of the generate MPL routines (see 14.1.7) or you can read in a DDT from its disk file (see 14.1.8).

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

14.1.6.1 Example Pascal Call for Multi-Point List Store

```
return_status := CM50_MPL_STORE
                 (mpl_name,
                  acidp_name,
                  %REF values_array1,
                  %REF values_array2,
                  %REF values_array3,
                  %REF values_array4,
                  store_array,
                  status_table,
                  cg_port_num);
```

14.1.6.2 Parameter Definitions for Multi-Point List Store

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (CM50_ACP_RUN)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (CM50_LCN_PART)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

mpl_name—The name of a Multi-Point List structure (declared as type cm50_idb_rec) defining the data to be stored.

acidp_name—A 16-character string (declared as type cm50_long_ACIDP) containing the name of an ACIDP. If the ACIDP is spaces, then the ACIDP currently connected to the ACP will control the data transfer. If an ACIDP is named, then the data access will be completed only if that ACIDP is in RUN state.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array that contains the data to be stored. The data type for each array must match the corresponding data type in the MPL definition. Each array should be declared as a CM50$VALUE_RECORD type.

store_array—The name of a shortword array (declared as type cm50$int300) that contains a control code entry for each value to be stored. These codes control what—if any—value is to be stored. The store code values are

0 – Store the value from the Values Array
1 – Store the bad value representation instead
2 – Do not store any value
16386 - Store IEEE negative infinity instead of Real value
16387 - Store IEEE positive infinity instead of Real value
16389 - Store IEEE negative zero instead of Real value

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

status_table—The name of a shortword array (declared as type cm50$int300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag in the list.. See Appendix A.1 for a listing of Data Access error/status codes.

cg_port_num—The name of a shortword (declared as type cm50$int2 with a value of 1-4) identifying the CG to be accessed.

14.1.7 Generate Multi-Point List

These routines generate an Internal data block for transfer arrays of up to four data types between the LCN and host computer. Internal data blocks are subject to exactly the same restrictions as DDTs (see Table 6-1).

A Multi-Point List may be generated from either a set of ID Block Arrays (such as those produced using the Convert Lists calls—see section 15.2.2), or a text file of type declarations and tag names, or an array of text entries.

NOTE

The arrays of internal point.parameter addresses need to be rebuilt and the program(s) using them need to be recompiled whenever the LCN database is changed in a significant manner, such as by the rebuild or deletion of data points referenced in the address array.

14.1.7.1 Example Pascal Calls to Generate Multi-Point Lists

To combine point lists, use:

```
return_status :=    CM50_MPL_GENLIST
                    (list_size,
                     id_block_arr1,
                     id_block_arr2,
                     id_block_arr3,
                     id_block_arr4,
                     mpl_name);
```

When the external ids are expressed as a Tag name list, use:

```
return_status :=    CM50_MPL_GENTAGS
                    (tagname_arr,
                     number_of_values,
                     mpl_name,
                     cg_port_num
                     return_arr);
```

When the external ids are contained in a Text file, use:

```
return_status :=    CM50_MPL_GENFILE
                    (tag_file,
                     mpl_name,
                     cg_port_num
                     return_arr);
```

14.1.7.2 Parameter Definitions for Generate Multi-Point Lists

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `return_array` status code for each returned value must be checked.

tagname_arr—The name of an array of up to 304 40-character strings (declared as cm50_tag_list_type) that contains the ASCII Tagname (formatted as Point.Param, optionally with the parameter index enclosed in parentheses) of the LCN entity for which the internal ID is to be obtained. All tags of the same data type must be grouped together and different data types must be separated by the reserved "tag" of: `**NEWΔTYPE=type`,
where Δ is a required space, and "type" (starting in position 12) is one of the following:

REAL	real number
INTE	integer
ASCI	24 character ASCII
ENUM	enumeration
ORDN	ordinal
PTID	internal entity id
EXID	external entity id
TIME	LCN internal time type
STRI	40 character string

If the first item in the array does not contain "**NEW TYPE=" in positions 1 through 11, then the first set of tags is assumed to identify Real numbers.

number_of_values —The name of a shortword (declared as type cm50$int2) specifying the number of tags defined in the tagname_arr. The maximum number of values is 304.

tag_file—The 80-character name (declared as cm50_file_name_type) of a text file whose content is a tagname_array, with each line containing either a valid tagname or a "**NEW TYPE=" tag as described above.

list_size —The name of an array of 4 shortwords (declared as type cm50_Ptid_vals) specifying the number of tags defined in each id_block_arr. The maximum number of values is 300.

id_block_arr*n* —(where **n** is 1 to 4) The name of a point list array (declared as cm50_point_list_array_type) which may combine up to 4 different data different types, with a maximum of 300 16-byte variables. If multiple data types are included, then all entries of the same type must be grouped together. The size of the point list must match that specified in list_size[**n**]. If there are fewer than 4 data types, the unused arguments may be omitted (but the correct number of commas is required).

mpl_name—The name of a Multi-Point List structure (declared as type cm50_idb_rec) where the generated definition is to be stored.

cg_port_num—The name of a shortword (declared as type cm50$INT2 with a value of 1-4) identifying the CG to be accessed.

return_arr—The name of an array of up to 304 integers (declared as cm50_return_arr_type) to receive the status of the conversion of each tag and data type declaration, including field type records. See Appendix A.2 for an explanation and a listing of all assigned return code values.

14.1.8 Read Multi-Point List

This routine reads an MPL from a disk file that has been created using either the DDT Build procedures or the Write Multi-Point List routine.

14.1.8.1 Example Pascal Calls to Read Multi-Point Lists

```
return_status :=     CM50_MPL_READ
                     (idb_file,
                      mpl_name);
```

14.1.8.2 Parameter Definitions for Read Multi-Point List

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

idb_file—The 80-character pathname (declared as cm50_file_name_type) of a file containing the Multi-Point List. To reference a DDT, use the pathname of `CM50$DDT:ddtname.II`. If no extension is specified, the default of .MPL will be used.

mpl_name—The name of a Multi-Point List structure (declared as type cm50_idb_rec) in memory.

14.1.9 Write Multi-Point List

This routine creates a disk file containing an MPL produced through the Generate Multi-Point List interface (section 14.1.7).

14.1.9.1 Example Pascal Calls to Write Multi-Point Lists

```
return_status :=     CM50_MPL_WRITE
                     (idb_file,
                      mpl_name);
```

14.1.9.2 Parameter Definitions for Write Multi-Point List

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

idb_file—The 80-character pathname (declared as cm50_file_name_type) of a file to contain the Multi-Point List. If a file by that name already exists, a new version will be created. By default, an extension of .MPL will be used. The use of .II as an extension is prohibited because that extension is reserved for DDTs. It is the user's responsibility to purge obsolete versions.

mpl_name—The name of an Multi-Point List structure (declared as type cm50_idb_rec) in memory.

14.1.10 Create Include File for Multi-Point List

This routine creates a disk file containing the text description of an MPL in a format suitable for use as an include file for a Pascal source program. The MPL should be previously produced through the Generate Multi-Point List interface (see heading 14.1.7).

14.1.10.1 Example Pascal Call to Generate a Multi-Point List Include File

```
return_status :=     CM50_MPL_GENINCL
                     (mpl_name,
                      text_file,
                      Language);
```

14.1.10.2 Parameter Definitions for Generate Multi-Point List Include File

return_status—The name of a integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

mpl_name—The name of a Multi-Point List structure (declared as type cm50_idb_rec) in memory. This should be declared as a record using the CM50_IDB_REC structure defined in the CM50_INCLUDE files.

text_file—The 80-character pathname (declared as cm50_file_name_type) of the include file to be written. If a file by that name already exists, a new version will be created. No default extension is provided. It is the user's responsibility to purge obsolete versions.

language— A single-character string (declared as type cm50_char) identifying the format of the include file:

'P' = Pascal
'C' = C
'F' = FORTRAN

Any other value will default to FORTRAN.

14.2 POINT LIST TRANSFERS

These routines enable you to address multiple points with a single call without the necessity to build DDT tables. In the place of a DDT reference, you will have to provide a pointer to an array of "internal" point.parameter addresses. These internal addresses can be obtained by conversion calls at program runtime (see heading 15.2), or in advance by creating an include file through the Utility MAKEINC (see heading 7.2).

14.2.1 Point List Get Values Interface

This function returns data values to up-to-300 points on the LCN without using DDT tables. The specification of which data is to be fetched and where it is to be stored is contained in the call.

Use of Internal Point-parameter IDs is required. Individual elements of parameter arrays can be specified by repeating the point.parameter address using a changed parameter index. The data type of the values is determined from the Internal Id of the first point in the list.

14.2.1.1 Example Pascal Call for Point List Get Values

```
return_status := CM50_GET_PT_LIST
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  values_array,
                  status_table,
                  number_of_values);
```

14.2.1.2 Parameter Definitions for Point List Get Values

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (CM50_ACP_RUN)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (CM50_LCN_PART)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

cg_port_num—The name of a shortword (declare as CM50_Uword) identifying the CG (1-4) to be accessed.

priority—The name of a shortword (declare as CM50_Uword) that contains the requested data-access priority:
1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

acidp_name—The name of a 16-character string (declare as CM50_long_ACIDP) that contains the name of an ACIDP. If the ACIDP name value is blank (all spaces), then the data is retrieved without any ACIDP controls. If an ACIDP is named, then the data access is completed only if that ACIDP is in RUN state.

point_list_array—The name of an array of 300 point addresses in internal format (declare as cm50_Point_List_array_type) from which the values are requested. See the Convert External to Internal ID functions (section 15.2) for additional information.

values_array—The name of a record structure (declare as CM50$Value_Record -- which has variant arrays for each data type) where the individual point values are to be stored.

status_table—The name of an array of 300 shortwords (declare as CM50_Intg_Array_type) where the value status for individual point values are to be stored. See Appendix A.1 for a listing of Data Access error/status codes.

number_of_values—The name of a shortword (declare as cm50_integer_2_type) that specifies the actual number of values (300 or less) to be processed.

14.2.2 Point List Get By Value Types

These functions are identical to the CM50_GET_PT_LIST function, except that the value type is part of the function name and the generic "values_array" argument is replaced by an array whose data type explicitly matches the specified data type.

These specific functions and their corresponding value arrays are described below. Refer to heading 14.2.1.2 for explanations of all of the other arguments.

14.2.2.1 Pascal Call for Point List Get Real Values

```
return_status := CM50_GET_REALNBR
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  real_values_array,
                  status_table,
                  number_of_values);
```

real_values_array—The name of an array of 300 Reals (declare as CM50_Real_Array_type) where the individual point values are to be stored.

14.2.2.2 Pascal Call for Point List Get Integer Values

```
return_status := CM50_GET_INTNBR
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  intg_values_array,
                  status_table,
                  number_of_values);
```

intg_values_array—The name of an array of 300 shortwords (declare as cm50_Intg_Array_type) where the individual point values are to be stored.

14.2.2.3 Pascal Call for Point List Get ASCII Values

```
return_status := CM50_GET_ASC24
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  asci_values_array,
                  status_table,
                  number_of_values);
```

ascii_values_array—The name of an array of 24-character strings (declare as cm50_ASCI_Array_type) where the individual point values are to be stored.

14.2.2.4 Pascal Call for Point List Get Enumerated Values

```
return_status := CM50_GET_ENUM
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  aenm_values_array,
                  status_table,
                  number_of_values);
```

aenm_values_array—The name of an array of 300 8-character strings (declare as cm50_Aenm_Array_type) where the individual point values are to be stored.

14.2.2.5 Pascal Call for Point List Get Ordinal Values

```
return_status := CM50_GET_ORD
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  oenm_values_array,
                  status_table,
                  number_of_values);
```

oenm_values_array—The name of an array of 300 shortwords (declare as cm50_Intg_Array_type) where the individual point values are to be stored.

14.2.2.6 Pascal Call for Point List Get Internal IDs

```
return_status := CM50_GET_PTID
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  ptid_values_array,
                  status_table,
                  number_of_values);
```

ptid_values_array—The name of an array of 300 64-bit internal entity ids (declare as cm50_Ptid_Array_type) where the individual point values are to be stored.

14.2.2.7 Pascal Call for Point List Get External IDs Values

```
return_status := CM50_GET_EXID
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  exid_values_array,
                  status_table,
                  number_of_values);
```

exid_values_array—The name of an array of 18-character strings (16 character point name followed by 2 character networked LCN identifier) where the individual point values are to be stored.

14.2.2.8 Pascal Call for Point List Get Time Values

```
return_status := CM50_GET_TIME
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  time_values_array,
                  status_table,
                  number_of_values);
```

time_values_array—The name of an array of 300 LCN time values (declare as cm50_Time_Array_type) where the individual point values are to be stored.

14.2.2.9 Pascal Call for Point List Get String Values

```
return_status := CM50_GET_STRI
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  stri_values_array,
                  status_table,
                  number_of_values);
```

stri_values_array—The name of an array of 40-character strings (declare as cm50_Stri_Array_type) where the individual point values are to be stored.

14.2.3 Point List Store Values Interface

This function exports data values to up to 300 points on the LCN without using DDT tables. The specification of which data is to be fetched and where it is to be stored is contained in the call.

Use of Internal Point-parameter IDs is required. Individual elements of parameter arrays can be specified by repeating the point.parameter address using a changed parameter index. The data type of the values is determined from the Internal Id of the first point in the list. Note: Entity ids can only be stored using their internal form.

14.2.3.1 Example Pascal Call for Point List Store Values

```
return_status := CM50_STORE_PT_LIST
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  values_array,
                  store_code_table,
                  status_table,
                  number_of_values);
```

14.2.3.2 Parameter Definitions for Array Store Values

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (CM50_ACP_RUN)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (CM50_LCN_PART)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

cg_port_num—The name of a shortword (declare as CM50_Uword) identifying the CG (1-4) to be accessed.

priority—The name of a shortword (declare as CM50_Uword) that contains the requested data-access priority:
1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

acidp_name—The name of a 16-character string (declare as CM50_long_ACIDP) that contains the name of an ACIDP. If the ACIDP name value is blank (all spaces), then the ACIDP currently connected to the ACP will control the data transfer. The data access is completed only if the named or implied ACIDP is in RUN state.

point_list_array—The name of an array of 300 point addresses in internal format (declare as cm50_Point_List_array_type) identifying where the values are to be stored. See the Convert External to Internal ID functions (section 15.2) for additional information.

values_array—The name of a record structure (declare as CM50$Value_Record -- which has variant arrays for each data type) where the individual point values are stored.

store_code_table—The name of an array of 300 shortwords (declare as cm50_Intg_Array_type) where the calling program has stored a control code for each value to be stored. These codes control what—if any—value is to be stored. The store code values are:

0 = Store the value from the Values Array
1 = Store the bad value representation for Real or ASCII only
2 = Do not store any value
16386 = Store IEEE negative infinity instead of Real value
16387 = Store IEEE positive infinity instead of Real value

status_table—The name of an array of 300 shortwords (declare as cm50_Intg_Array_type) where the value status for individual point values are to be stored. See Appendix A.1 for a listing of Data Access error/status codes.

number_of_values—The name of a shortword (declare as cm50_Uword) that specifies the actual number of values (300 or less) to be processed.

14.2.4 Point List Store By Value Type

These functions are identical to the CM50_STORE_PT_LIST function, except that the value type is part of the function name and the generic "values_array" argument is replaced by an array whose data type explicitly matches the specified data type.

These specific functions and their corresponding value arrays are described below. Refer to heading 14.2.3.2 for explanations of all of the other arguments.

14.2.4.1 Pascal Call for Point List Store Real Values

```
return_status := CM50_STORE_REALNBR
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  real_values_array,
                  store_code_table,
                  status_table,
                  number_of_values);
```

real_values_array—The name of an array of 300 Reals (declare as cm50_Real_Array_type) containing the values to be stored.

14.2.4.2 Pascal Call for Point List Store Integer Values

```
return_status := CM50_STORE_INTNBR
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  intg_values_array,
                  store_code_table,
                  status_table,
                  number_of_values);
```

intg_values_array—The name of an array of 300 shortwords (declare as cm50_Intg_Array_type) containing the values to be stored.

14.2.4.3 Pascal Call for Point List Store ASCII Values

```
return_status := CM50_STORE_ASC24
                 (cg_port_num,
                  priority,
                  acidp_name),
                  point_list_array,
                  asci_values_array,
                  store_code_table,
                  status_table,
                  number_of_values);
```

asci_values_array—The name of an array of 300 24-character strings (declare as cm50_ASCI_Array_type) containing the values to be stored.

14.2.4.4 Pascal Call for Point List Store Enumerated Values

```
return_status := CM50_STORE_ENUM
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  aenm_values_array,
                  store_code_table,
                  status_table,
                  number_of_values);
```

aenm_values_array—The name of an array of 300 8-character strings (declare as cm50_Aenm_Array_type) containing the values to be stored.

14.2.4.5 Pascal Call for Point List Store Ordinal Values

```
return_status := CM50_STORE_ORD
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  oenm_values_array,
                  store_code_table,
                  status_table,
                  number_of_values);
```

oenm_values_array—The name of an array of 300 shortwords (declare as cm50_Intg_Array_type) containing the values to be stored.

14.2.4.6 Pascal Call for Point List Store Internal IDs

```
return_status := CM50_STORE_PTID
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  ptid_values_array,
                  store_code_table,
                  status_table,
                  number_of_values);
```

ptid_values_array—The name of an array of 300 internal point ids (declare as cm50_Ptid_Array_type) containing the values to be stored.

14.2.4.7 Pascal Call for Point List Store Time Values

```
return_status := CM50_STORE_TIME
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  time_values_array,
                  store_code_table,
                  status_table,
                  number_of_values);
```

time_values_array—The name of an array of 300 LCN time values (declare as cm50_Time_Array_type) containing the values to be stored.

14.2.4.8 Pascal Call for Point List Store String Values

```
return_status := CM50_STORE_STRI
                 (cg_port_num,
                  priority,
                  acidp_name,
                  point_list_array,
                  stri_values_array,
                  store_code_table,
                  status_table,
                  number_of_values);
```

stri_values_array—The name of an array of 300 40-character strings (declare as cm50_Stri_Array_type) containing the values to be stored.

14.3 SINGLE POINT DATA TRANSFERS

The interface routines in this group Get or Store values from or to one named point.parameter (or parameter array) at a time. For parameter arrays, up-to the whole array is accessed. The External ID version of Get Single Point is also used to get LCN date and time.

14.3.1 Single Point Get Data (External ID) Interface

This routine fetches data for a single point from a specified CG or elsewhere on its LCN. The specification of which data is to be fetched and where it is to be stored is contained in the call. For parameter arrays, either a single element, the whole array, or an array subset starting with the first element can be specified. The point may be identified by either a combination of point and parameter names or by a single tag name.

14.3.1.1 Example Pascal Calls for Single Point Get

Using point and parameter names as separate variables:

```
return_status := CM50_GET_ID
                 (entity,
                  param,
                  param_ix,
                  val_loc,
                  val_st,
                  val_typ,
                  cg_port_num);
```

Using a complete tag name:

```
return_status := CM50_GET_TAG
                 (tag_name,
                  val_loc,
                  val_st,
                  val_typ,
                  cg_port_num);
```

14.3.1.2 Parameter Definitions for Single Point Get

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000051 (CM50_LCN_PART)—the `val_st` status code for each returned value must be checked.

215000322 (CM50_ACC_SIZE)—the array size specified by `param_ix` is larger than the actual size.

tag_name—The name of a 40-character string (declared as CM50_tag_name_type) that identifies the LCN value to be retrieved. The tag name is formatted as "point.param (param_ix)".

entity—The name of a 20-character string (declared as CM50_entity_name_type) that contains the ASCII Point ID. It should contain a point name of up to 16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter for Network Gateway routing.

param—The name of an 8-character string (declared as type CM50_ASCII_param_arr) that contains the ASCII name of a parameter (or parameter array) from which the value(s) is retrieved.

param_ix—The name of a shortword (declare as cm50$int2) that contains the parameter index. Use of this value is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, 5, 13, 15, 17 or 19, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of `param_ix` is used as an index and must be greater than zero. If the parameter being accessed is not an array type, `param_ix` must be zero.

When `val_typ` is 7, 8, 9, 10, 14, 16, 18 or 20, a whole array (or a subset of the array starting with the first element) is to be accessed and `param_ix` is used to specify the number of elements to be accessed—If `param_ix` is smaller than the actual array size, only that number of elements is returned; if it is larger than the actual array size, no elements are returned and the return_status value is 215000322.

val_loc—The name of a program variable (declared as a CM50$value_record) where the value(s) are to be stored. The CM50$value_record has variants that match each value type:

val_typ	variant	data type
1	.a	Real (32 bits)
2	.b	Integer (shortword)
3	.c	ASCII values (24-character string)
4	.d	Enumeration (8-character string)
5	.e	Ordinal (shortword)
6	.f	External Time (18-character string)
7	.g	Array [1..n] of Real
8	.h	Array [1..n] of Integer
9	.i	Array [1..n] of Enumeration
10	.j	Array [1..n] of Ordinals
13	.m	Internal_id (CM50_Ptid_vals)
14	.n	Array [1..n] Internal_id
15	.o	External_id (18-character string))
16	.p	Array [1..n] of External_id
17	.q	Internal Time (CM50_Time_Vals record)
18	.r	Array [1..n] of Internal Time
19	.s	String (40 characters)
20	.t	Array [1..n] of String

val_st—The name of a shortword (declared as CM50$int2) where point-related status information is to be stored. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (CM50_LCN_PART). When `val_typ` specifies an array, `val_st` refers to status of the whole array.

val_typ—The name of a shortword that contains a number that designates value type of the accessed parameters as listed for **val_loc**.

cg_port_num—The name of a shortword (declared as CM50$int2) identifying the CG to be accessed.

14.3.2 Single Point Store Data (External ID) Interface

This routine stores data to a single point in a specified CG or elsewhere on its LCN. The specification of where the data is to be found and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

To use this call the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

14.3.2.1 Example Pascal Calls for Single Point Store

Using point and parameter names as separate variables:

```
return_status := CM50_STORE_ID
                 (entity,
                  param,
                  param_ix,
                  val_loc,
                  val_typ,
                  store_cd,
                  store_st);
```

Using a complete tag name:

```
return_status := CM50_STORE_TAG
                 (tag_name,
                  val_loc,
                  val_typ,
                  store_cd,
                  store_st);
```

14.3.2.2 Parameter Definitions for Single Point Store

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `store_st` status code for each returned value must be checked.

tag_name—The name of a 40-character string (declared as CM50_tag_name_type) that identifies the LCN value(s) to be stored. The tag name is formatted as "point.param (param_ix)".

entity—The name of a 20-character string (declared as CM50_Entity_name_type) that contains the ASCII Point ID. It should contain a point name of up-to-16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter for Network Gateway routing.

param—The name of 8-character string (declared as CM50_ASCII_Param_arr) that contains the ASCII parameter name for the point.parameter where the value is to be stored.

param_ix—The name of a shortword (declared as CM50$int2) containing the parameter index. Use of this value is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, 5, 13, 17, or 19, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of `param_ix` is used as an index and must be greater than zero. If the parameter being accessed is not an array type, `param_ix` must be zero.

When `val_typ` is 7, 8, 9, 10, 14, 18, or 20, a whole array is to be accessed and `param_ix` is used to specify the number of array elements—If `param_ix` does not match the actual array size, no elements are stored and `return_status` value is 5 with a `store_st` indicating an invalid array size.

val_loc—The name of a program variable (declared as a CM50$value_record) containing the value(s) to be stored. The CM50$value_record has variants that match each value type:

val_typ	variant	data type
1	.a	Real (32 bits)
2	.b	Integer (shortword)
3	.c	ASCII values (24-character string)
4	.d	Enumeration (8-character string)
5	.e	Ordinal (shortword)
6	.f	External Time (18-character string)
7	.g	Array [1..n] of Real
8	.h	Array [1..n] of Integer
9	.i	Array [1..n] of Enumeration
10	.j	Array [1..n] of Ordinals
13	.m	Internal_id (CM50_Ptid_vals)
14	.n	Array [1..n] Internal_id
17	.q	Internal Time (CM50_Time_Vals record)
18	.r	Array [1..n] of Internal Time
19	.s	String (40 characters)
20	.t	Array [1..n] of String

val_typ—The name of a shortword (declared as CM50$int2) that contains a number that designates value type as listed above.

store_cd—Name of a shortword (declared as CM50$int2) that contains a code that allows the substitution of a bad value representation in place of the provided value(s). The store code values are:

0 = Store the data value(s) provided
1 = Store the bad value representation instead

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

store_st—The name of a shortword (declared as CM50$int2) to contain point-related store status information on completion. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (CM50_LCN_PART). When the `val_typ` is an array, `store_st` refers to status of the whole array.

14.3.3 Single Point Get Data (Internal ID) Interface

This routine fetches data for a single point from the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the Entity Name conversion functions -- section 15.2.1) reduces the overhead required for repetitive single-point requests.

The specification of which data is to be fetched and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

14.3.3.1 Example Pascal Call for Single Point Get

```
return_status := CM50_GETPT_ID
                 (id_block,
                  %REF val_loc,
                  val_st,
                  cg_port_num)
```

14.3.3.2 Parameter Definitions for Single Point Get

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `val_st` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (declared as CM50$int8) that contains the internal ID data block obtained by a previous Convert External to Internal ID call. When the data is of array type, that call returns the array size in word 7 of the ID block. Thus, if you wish to get less than the entire array you can change the parameter qualifier in the seventh word of the ID block to be smaller than the actual array size. Do not change any other words in the ID block. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

val_loc—The name of a program variable (declared as a CM50$value_record) where the value(s) are to be stored. The CM50$value_record has variants that match each value type:

val_typ	variant	data type
1	.a	Real (32 bits)
2	.b	Integer (shortword)
3	.c	ASCII values (24-character string)
4	.d	Enumeration (8-character string)
5	.e	Ordinal (shortword)
6	.f	External Time (18-character string)
7	.g	Array [1..n] of Real
8	.h	Array [1..n] of Integer
9	.i	Array [1..n] of Enumeration
10	.j	Array [1..n] of Ordinals
13	.m	Internal_id (CM50_Ptid_vals)
14	.n	Array [1..n] Internal_id
15	.o	External_id (18-character string))
16	.p	Array [1..n] of External_id
17	.q	Internal Time (CM50_Time_Vals record)
18	.r	Array [1..n] of Internal Time
19	.s	String (40 characters)
20	.t	Array [1..n] of String

val_st—The name of a shortword (declared as CM50$int2) where point-related status information is to be stored. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (215000051). See Appendix A.1 for a listing of Data-Access error/status codes. When the `val_typ` specifies an array, `val_st` refers to status of the whole array.

cg_port_num—The name of a shortword (declared as CM50$int2) identifying the CG (1-4) to be accessed.

14.3.4 Single Point Store Data (Internal ID) Interface

This routine stores data to a single point in the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the Entity Name conversion functions -- section 15.2.1) reduces the overhead required for repetitive single-point requests.

The specification of where the data is found and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

To use this function the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

14.3.4.1 Example Pascal Call for Single Point Store

```
return_status := CM50_STOREPT_ID
                 (id_block,
                  %REF val_loc,
                  store_cd,
                  store_st);
```

14.3.4.2 Parameter Definitions for Single Point Store

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `store_st` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (declared as CM50$int8) that contains the internal ID data block obtained by a previous Convert External to Internal ID call. Do not change any words in the ID block. If the array size is changed, the array is not stored and the `return_status` value is 215000051, with a `store_st` that indicates an invalid array size. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

val_loc—The name of a program variable (declared as a CM50$value_record) containing the value(s) to be stored. The CM50$value_record has variants that match each value type:

val_typ	variant	data type
1	.a	Real (32 bits)
2	.b	Integer (shortword)
3	.c	ASCII values (24-character string)
4	.d	Enumeration (8-character string)
5	.e	Ordinal (shortword)
6	.f	External Time (18-character string)
7	.g	Array [1..n] of Real
8	.h	Array [1..n] of Integer
9	.i	Array [1..n] of Enumeration
10	.j	Array [1..n] of Ordinals
13	.m	Internal_id (CM50_Ptid_vals)
14	.n	Array [1..n] Internal_id
17	.q	Internal Time (CM50_Time_Vals record)
18	.r	Array [1..n] of Internal Time
19	.s	String (40 characters)
20	.t	Array [1..n] of String

store_cd—The name of a shortword (declared as CM50$int2) that contains a code that allows the substitution of a bad value representation in place of the provided value(s). The store code values are

0 = Store the data value(s) provided
1 = Store the bad value representation instead

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

store_st—The name of a shortword (declared as CM50$int2) where point-related status information is to be stored. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (215000051). See Appendix A.1 for a listing of Data-Access error/status codes. When the `val_typ` specifies an array, `store_st` refers to status of the whole array.

14.3.5 Get LCN Clock Value Interface

The current date and time as kept by the LCN, can be obtained in either internal or ASCII format. The internal format is a 4-byte integer count of the number of seconds since January 1, 1979. The ASCII format is MM/DD/YYΔHH:MM:SSΔ (where Δ is used to indicate a space).

14.3.5.1 Example Pascal Calls to Get the LCN Clock

Internal Time Format:

```
return_status :=   CM50_TIMNOW_LCN
                   (Integer_Clock,
                    cg_port_num);
```

ASCII Time Format:

```
return_status :=   CM50_TIMNOW_ASC
                   (ASCII_Clock,
                    cg_port_num);
```

14.3.5.2 Parameter Definitions for Get LCN Clock

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

Integer_clock—The name of an integer where the clock value, in seconds, is to be returned.

ASCII_clock—The name of an 18-character string (declared as type CM50$Time_arr) where the clock value, formatted as 'MM/DD/YY hh:mm:ss ', is to be returned.

cg_port_num—The name of a shortword (declare as CM50$INT2, with a value of 1-4) identifying the CG to be accessed.

14.4 RAW DATA TRANSFERS

The interface routines in this group get, store, and convert **only** LCN **Real data arrays** in LCN format. Each request works only with a single data point's parameter array. These functions allow you to pass Real data arrays from one LCN to another without needing to go through the LCN/VAX data conversions.

14.4.1 Get Raw Data Interface

This function fetches data for a single point from the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the Convert External to Internal ID interface, see 15.2.1) is required.

The specification of which data is to be fetched and where it is to be stored is contained in the call.

14.4.1.1 Example Pascal Call for Get Raw Data

```
return_status := CM50_SPGRAW
                 (id_block,
                  value_loc,
                  priority,
                  value_status,
                  cg_port_num);
```

14.4.1.2 Parameter Definitions for Get Raw Data

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `value_status` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (declare as type CM50$int8) that contains the internal ID data block obtained by a previous Convert External to Internal ID request. When the conversion returns the array size in word 7 of the ID block. Thus, if you wish to get less than the entire array you can change the parameter qualifier in the seventh word of the ID block to be smaller than the actual array size. Do not change any other words in the ID block. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

value_loc—The name of a Real array (declare as type CM50$real1000) where the values are to be stored. The id_block should identify the value type as 7 (Real array).

priority—The name of a shortword (declare as type CM50$int2) that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

value_status—The name of a shortword (declare as type CM50$int2) where point-related status information is to be stored. This value is meaningful only when the return_status value indicates normal, complete with errors, or array-size error. See Appendix A.1 for a listing of Data Access error/status codes. The value refers to status of the whole array.

cg_port_num—The name of a shortword (declare as type CM50$int2) identifying the CG (1-4) to be accessed.

14.4.2 Store Raw Data Interface

This function stores data to a single point in the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the Convert External to Internal ID interface, see 15.2.1) is required.

The specification of where the data is found and where it is to be stored is contained in the call.

To use this function the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

14.4.2.1 Example Pascal Call for Store Raw Data

```
return_status := CM50_SPSRAW
                 (id_block,
                  value_loc,
                  priority,
                  store_code,
                  value_status,
                  cg_port_num);
```

14.4.2.2 Parameter Definitions for Store Raw Data

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `value_status` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (declare as type CM50$int8) that contains the internal ID data block obtained by a previous Convert External to Internal ID call. Do not change any words in the ID block. If the array size is changed, the array is not stored and the return_status value is 5 with a value_status that indicates an invalid array size. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

value_loc—The name of a Real array (declare as type CM50$real1000) that contains the value or values to be stored. The id_block should identify the value type as 7 (Real array).

priority—The name of a shortword (declare as type CM50$int2) that contains the requested data-access priority:

1 = High priority (provided for control operations)
2 = Low priority (provided for noncontrol operations)

store_code—The name of a shortword (declare as type CM50$int2) that contains a code that allows the substitution of a bad value representation in place of the provided value(s):

0 = Store the data value(s) provided
1 = Store the bad value representation (NaN) instead

value_status—The name of a shortword (declare as type CM50$int2) where point-related status information is to be stored. This value is meaningful only when the return_status value indicates normal or complete with errors. See Appendix A.1 for a listing of Data-Access error/status codes. The value refers to status of the whole array.

cg_port_num—The name of a shortword (declare as type CM50$int2) identifying the CG (1-4) to be accessed.

14.4.3 Convert Raw Data

This function converts the elements of a Real array from LCN format to VAX format.

14.4.3.1 Example Pascal Call for Convert Raw Data

```
return_status := CM50_SPCRAW
                 (id_block,
                  raw_val_loc,
                  vax_val_loc,
                  value_type,
                  convert_status);
```

14.4.3.2 Parameter Definitions for Convert Raw Data

return_status—The name of a longword to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `convert_status` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (declare as type CM50$int8) that contains the internal ID data block obtained by a previous Convert External to Internal ID call. Do not change any words in the ID block. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

raw_val_loc—The name of a Real array (declare as type CM50$real1000) that contains previously obtained raw values that are to be converted from LCN format.

vax_val_loc—The name of a Real array (declare as type CM50$real1000) to contain the converted values.

value_type—The name of a shortword value (declare as type CM50$int2) that must =7 (Real array).

convert_status—The name of a shortword array (declare as type CM50$int1000) where the request-completion status for each data array element is to be stored. Value meanings are

0 = Normal return; this element was converted successfully
1 = Unable to convert this element to VAX format
2 = Bad value substitution was done on this element

14.5 HISTORY DATA TRANSFERS

The interface routines in this group get previously stored averages or 1-minute snapshot data from a History Module on the LCN. The data may be requested using a DDT, Internal Data Block or the internal address of a single tag. The History calls provide for concurrent Get History requests by up-to-four application programs. A fifth request is rejected with a queue-full status return.

14.5.1 Selecting Records From the History Module

The History Module uses a specialized set of circular files to hold historized values collected from data points on the LCN. Effective use of the CM50S history functions requires an understanding of data organization on the History Module.

14.5.1.1 Relative and Absolute Time References

The History Module may be searched using either Relative or Absolute time references. Relative references request data based on a number of records offset from the current value. Absolute Time reference request data for all records whose timestamps fall within a specified Date/Time interval.

For **Absolute Time** references, the Begin Date/Time specifies the timestamp of the most recent value to be retrieved and the End Date/Time specifies the timestamp of the oldest value to be retrieved. If a seasonal time change has occurred during a specified Absolute History interval, the number of samples returned can differ from the expected number of samples. For example, if it is desired to obtain a day's worth of hourly averages (24) and a forward time change of one hour has occurred, 23 samples are returned. If the time change is in the backward direction, 25 samples are returned.

Relative requests are based on beginning and ending offsets which are counts of records back from the current time. The direction of search can be either forward (oldest to newest data) or backwards (newest to oldest data); however, a forward search requires at least twice as long to execute. To execute a backward search, set the starting offset value less-than or equal-to the ending offset value. The number of samples returned is calculated as the positive difference between the starting offset and the ending offset plus one. If this difference exceeds 262, the request is truncated at 262 samples. The number of samples returned by a Relative History request is immune to time changes.

Offset values less than one have special meanings. When the starting or ending offset value is zero (i.e., current LCN time) in the case of averages, the first sample returned is the current running average for the period. A starting offset of -1 has special meaning in the cases of snapshots and user averages. In those cases only, LCN time is rounded to the beginning of the last hour. This permits an ACP to be sure of obtaining the last full hour of snapshots or user averages. In calculating the number of samples returned, a -1 is treated as an offset of 0 and its number of samples and direction of search follows those rules. An ending offset of -1 for snapshots and user averages means the search direction is forward and the ending time is on the hour starting "n" units back from current time.

The following table summarizes results of combinations of starting and ending offsets for Relative History requests with numbers of samples returned and reasons for zero sample returns.

History Type	Starting Offset	Ending Offset	Number of Samples	Direction of Search	Partial First Sample for Averages?
any	0	0	1	Backward	yes
any	1	1	1	Backward	no
any	2	3	2	Backward	no
any	3	2	2	Forward	no
any	0	300	262	Backward	yes
0,5	3	-1	4	Forward	no
1 to 4	3	-1	0	Error, end offset invalid	
0,5	-1	3	4	Backward	no
0,5	-1	-3	0	Error, end offset invalid	
1 to 4	-1	-3	0	Error, begin/end offset invalid	

14.5.1.2 Number of Values Retrieved in a Single Call

The number of values that can be obtained from the History Module for each point is limited both by the size of the buffer used to transfer the values and by the History type. The maximum number of values for monthly averages is 12, and for shift averages is 21. The maximum for user averages is configuration dependent, but will not exceed the number of values shown below for hourly averages. The other maximums are shown in the following table.

Number of Points in DDT or List	Maximum Snapshots	Maximum Hourly Averages	Maximum Daily Averages
1-3	262	168	31
4	262	149	31
5	238	119	31
6	198	99	31
7	170	85	31
8	149	74	31
9	132	66	31
10	119	59	31
11	108	54	31
12	99	49	31
13	91	45	31
14	85	42	31
15	79	39	31
16	74	37	31
17	69	34	31
18	66	33	31
19	62	31	31
20	59	29	29
21	56	28	28
22	53	26	27
23	51	25	25
24	49	24	24

14.5.2 Get History Snapshots (Relative Time)

These routines are used to fetch history snapshots from the HM, using a relative offset from current LCN time.

14.5.2.1 Example Pascal Calls for Get History Snapshots (Relative Time)

for standard 1-minute snapshots:

```
return_status := CM50_DDTHIS_SNAP
                 (ddt_name,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_offset,
                  end_offset);
```

for fast (5, 10 or 20 second) snapshots:

```
return_status := CM50_DDTHIS_FAST
                 (ddt_name,
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_offset,
                  end_offset);
```

for Multi-Point Lists (instead of DDT):

```
return_status := CM50_MPL_SNAP
                 (mpl_name,
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_offset,
                  end_offset,
                  cg_port_num);
```

for a single data point.parameter:

```
return_status := CM50_PTHIS_SNAP
                 (id_block,
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_offset,
                  end_offset,
                  cg_port_num);
```

14.5.2.2 Parameter Definitions for Get History Snapshots (Relative Time)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (declared as type CM50$DDT_string) that contains the ASCII name of the DDT to be used.

mpl_name—The name of an Multi-Point List structure (declared as type CM50_idb_rec) defining the data to be retrieved.

id_block—The name of a 16-byte variable (declared as type CM50$int8) containing the internal ID for an LCN tag. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

sample_rate —The name of a shortword (declared as type CM50$int2) identifying the number of snapshots to be returned for each minute. This value does not have to match the rate at which snapshots are historized. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

Note: Retrieval of more than 1 snapshot per minute is only supported by LCN release 400.

number_of_values—The name of a shortword (declared as type CM50$int2) that specifies the maximum number of history values (1..262) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between `begin_offset` and `end_offset`, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with status_table entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points will be lost if the `number_of_values` times the number of points is greater than 1197.

real_values_array—The name of a Real array (declared as type CM50$snap_array) where the history data is to be stored.

status_table—The name of a shortword array (declared as type CM50$hist_array) to contain the value status for each returned snapshot. If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: value returned is analog (real) data
1 = Nonstandard: not applicable
2 = Digital Value: value returned is the Real equivalent of an ordinal value for a self-defined enumeration
3-4 = not used
5 = Time Change: a time change occurred and data for one minute is missing; value field contains NaN
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: History collection was not enabled; value field contains NaN
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: not applicable
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX:

CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an integer array (declared as type CM50$time_array) that will contain the time stamp in seconds for each returned snapshot. See heading 15.3 for time-stamp conversions.

begin_offset—The name of a shortword (declared as type CM50$int2) that indicates a relative offset in minutes from current LCN time that represents the starting period for which history is to be fetched.

end_offset—The name of a shortword (declared as type CM50$int2) that indicates a relative offset in minutes from the current LCN time representing the ending period for which history is to be fetched.

cg_port_num—The name of a shortword (declared as type CM50$int2, with a value of 1-4) identifying the CG to be accessed.

14.5.3 Get History Snapshots (Absolute Times)

These routines are used to fetch history snapshots from the HM, using absolute begin and end times. Separate calls are provided for snapshot and averages histories.

If a seasonal time change has occurred during a specified Absolute History interval, the number of samples returned can differ from the expected number of samples. For example, if it is desired to obtain a day's worth of hourly averages (24) and a forward time change of one hour has occurred, 23 samples are returned. If the time change is in the backward direction, 25 samples are returned.

14.5.3.1 Example Pascal call for Get History Snapshots (Absolute Times)

for standard 1-minute snapshots:

```
return_status := CM50_DDTHIS_SNAPT
                 (ddt_name,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_date_time,
                  end_date_time);
```

for fast (5, 10 or 20 second) snapshots:

```
return_status := CM50_DDTHIS_FASTT
                 (ddt_name,
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_date_time,
                  end_date_time);
```

for Multi-Point Lists (instead of DDT):

```
return_status := CM50_MPL_SNAPT
                 (mpl_name,
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_date_time,
                  end_date_time,
                  cg_port_num);
```

for a single data point.parameter:

```
return_status := CM50_PTHIS_SNAPT
                 (id_block,
                  sample_rate,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_date_time,
                  end_date_time,
                  cg_port_num);
```

14.5.3.2 Parameter Definitions for Get History Snapshots (Absolute Times)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (declared as type CM50$DDT_string) that contains the ASCII name of the DDT to be used.

mpl_name—The name of an Multi-Point List structure (declared as type CM50_idb_rec) defining the data to be retrieved.

id_block—The name of a 16-byte variable (declared as type CM50$int8) containing the internal ID for an LCN tag. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

sample_rate —The name of a shortword (declared as type CM50$int2) identifying the number of snapshots to be returned for each minute. This value does not have to match the rate at which snapshots are historized. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

Note: Retrieval of more than 1 snapshot per minute is only supported by LCN release 400.

number_of_values—The name of a shortword (declared as type CM50$int2) that specifies the maximum number of history values (1..261) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between begin and end times, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with `status_table` entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some points are lost if the `number_of_values` times (1 + the number of points) is greater than 1197.

real_values_array—The name of a Real array (declared as type CM50$snap_array) where the history data is to be stored.

status_table—The name of a shortword array (declared as type CM50$hist_array) to contain the value status for each returned snapshot. If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: value returned is analog (real) data
1 = Nonstandard: not applicable
2 = Digital Value: value returned is the Real equivalent of an ordinal value for a self-defined enumeration
3-4 = not used
5 = Time Change: a time change occurred and data for one minute is missing; value field contains NaN
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: History collection was not enabled; value field contains NaN
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: not applicable
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX:

CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an integer array (declared as type CM50$time_array) that will contain the time stamp in seconds for each returned snapshot. See heading 15.3 for time-stamp conversions.

begin_date_time—The name of a 14-character string (declared as CM50_lcn_asctim_type) in the format MM/DD/YYΔHH:MM (where Δ indicates a blank character) specifying the date and time for the most recent record to be fetched from the History Module.

The value entered for this parameter should be some time after the desired time period begins, but before the next time period's collection time. For example, begin_date_time for an 11:00 hourly average should be set to any time from 11:01 to 11:59.

end_date_time—The name of a 14-character string (declared as CM50_lcn_asctim) in the format MM/DD/YYΔHH:MM, specifying the date and time for the oldest record to be fetched from the History Module. The end_date_time must be earlier than begin_date_time.

The value entered for this parameter should be some time after the previous period's collection, but before the desired time period begins. For example, to get an hourly average at 11:00, end_date_time should be set any time between 10:01 and 10:59.

cg_port_num—The name of a shortword (declared as type CM50$int2 with a value of 1-4) identifying the CG to be accessed.

14.5.4 Get History Averages (Relative Times)

These calls return the average, minimum and maximum values of a point for specified time periods.

14.5.4.1 Example Pascal call for Get History Averages (Relative Times)

```
return_status := CM50_DDTHIS_AVER
                 (ddt_name,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset,
                  history_type);
```

for Multi-Point Lists (instead of DDT):

```
return_status := CM50_MPLHIS_AVER
                 (mpl_name,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset,
                  history_type,
                  cg_port_num);
```

for a single data point.parameter:

```
return_status := CM50_PTHIS_AVER
                 (id_block,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset,
                  history_type,
                  cg_port_num);
```

14.5.4.2 Parameter Definitions for Get History Averages (Relative Times)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character array (declared as type CM50$DDT_string) that contains the ASCII name of the DDT to be used.

mpl_name—The name of an Multi-Point List structure (declared as type CM50_idb_rec) defining the data to be retrieved.

id_block—The name of a 16-byte variable (declared as type CM50$int8) containing the internal ID for an LCN tag.. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of a shortword (declared as type CM50$int2) that specifies the maximum number of history items (1..262) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between `begin_offset` and `end_offset`, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with `status_table` entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times the number of points is greater than 598.

real_values_array—The name of a Real array (declared as type CM50$aver_array) where the history data is to be stored.

status_table—The name of a shortword array (declared as type CM50$hist_array) to contain the value status for each returned snapshot. If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX
CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an integer array (declared as type CM50$time_array) that will contain the time stamp in seconds for each returned average. See heading 15.3 for time-stamp conversions.

max_array—The name of a real array (declared as type CM50$aver_array) that will contain the maximum process value recorded in the averaged period. Due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of a real array (declared as type CM50$aver_array) that will contain the minimum process value recorded in the averaged period. Due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an array of unsigned shortwords (declared as type CM50_num_samples_array_type) that will contain the number of samples used in calculating each returned average value.

begin_offset—The name of a shortword (declared as type CM50$int2) that indicates a relative offset from current LCN time that represents the first history record to be fetched.

end_offset—The name of a shortword (declared as type CM50$int2) that indicates a relative offset from the current LCN time representing the last history record to be fetched.

history_type—The name of a shortword (declared as type CM50$int2) that contains the number specifying the type of average requested. The available types and maximum number of records on the History Module for each are:

1 = Hourly (168 records)
2 = Shift (21 records)
3 = Daily (31 records)
4 = Monthly (12 records)
5 = User (configuration dependent)

cg_port_num—The name of a shortword (declared as type CM50$int2 with a value of 1-4) identifying the CG to be accessed.

14.5.5 Get History Averages (Absolute Times)

These calls return the average, minimum and maximum values of a point for specified time periods.

14.5.5.1 Example Pascal call for Get History Averages (Absolute Times)

```
return_status := CM50_DDTHIS_AVERT
                 (ddt_name,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time,
                  history_type);
```

for Multi-Point Lists (instead of DDT):

```
return_status := CM50_MPLHIS_AVERT
                 (mpl,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time,
                  history_type,
                  cg_port_num);
```

for single point requests:

```
return_status := CM50_PTHIS_AVERT
                 (id_block,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time,
                  history_type,
                  cg_port_num);
```

14.5.5.2 Parameter Definitions for Get History Averages (Absolute Times)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (declared as type CM50$DDT_string) that contains the ASCII name of the DDT to be used.

mpl —The name of an Multi-Point List structure (declared as type CM50_idb_rec) defining the data to be retrieved.

id_block—The name of a 16-byte variable (declared as type CM50$int8) containing the internal ID for an LCN tag. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of a shortword (declared as type CM50$int2) that specifies the maximum number of history items (1..261) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between begin and end times, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with status_table entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times (1 + the number of points) is greater than 598.

real_values_array—The name of a Real array (declared as type CM50$aver_array) where the history data is to be stored.

status_table—The name of a shortword array (declared as type CM50$hist_array) to contain the value status for each returned snapshot. If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX:
CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an integer array (declared as type CM50$time_array) to receive the time stamp in seconds for each returned average. See heading 15.3 for time-stamp conversions.

max_array—The name of a real array (declared as type CM50$aver_array) that will contain the maximum process value recorded in the averaged period. **Note:** Due to the data compression algorithm on the History module, there may be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of a real array (declared as type CM50$aver_array) that will contain the minimum process value recorded in the averaged period. **Note:** Due to the data compression algorithm on the History module, there may be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an array of unsigned shortwords (declared as type CM50_num_samples_array_type) that will contain the number of samples used in calculating each returned average value.

begin_date_time—The name of a 14-character string (declared as CM50_lcn_asctim_type) in the format MM/DD/YYΔHH:MM (where Δ indicates a blank character) specifying the date and time for the most recent record to be fetched from the History Module.

The value entered for this parameter should be some time after the desired time period begins, but before the next time period's collection time. For example, begin_date_time for an 11:00 hourly average should be set to any time from 11:01 to 11:59.

end_date_time—The name of a 14-character string (declared as CM50_lcn_asctim_type) in the format MM/DD/YYΔHH:MM, specifying the date and time for the oldest record to be fetched from the History Module. The end_date_time must be earlier than the begin_date_time.

The value entered for this parameter should be some time after the previous period's collection, but before the desired time period begins. For example, to get an hourly average at 11:00, end_date_time should be set any time between 10:01 and 10:59.

history_type—The name of a shortword (declared as type CM50$int2) that contains the number specifying the type of average requested. The available types and maximum time retained on the History Module for each are:

1 = Hourly (7 days)
2 = Shift (7 days)
3 = Daily (31 days)
4 = Monthly (1 year)
5 = User (8 hours to 7 days, depending on configuration)

cg_port_num—The name of a shortword (declared as type CM50$int2 with a value of 1-4) identifying the CG to be accessed.

14.5.6 Get Monthly Averages (Relative Times)

When a point is historized more often than once per minute, it is possible for the number of samples taken during a month to exceed the capacity of a 16-bit integer. This call provides a 32-bit integer count of the number of samples in a monthly average using relative time.

NOTE

Retrieval of monthly averages using this call is only supported by LCN release 400 or later.

14.5.6.1 Example Pascal call for Get Monthly Averages (Relative Times)

```
return_status := CM50_DDTHIS_MNTH
                 (ddt_name,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset);
```

for Multi-Point Lists (instead of DDT):

```
return_status := CM50_MPLHIS_MNTH
                 (mpl_name,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset,
                  cg_port_num);
```

for a single data point.parameter:

```
return_status := CM50_PTHIS_MNTH
                 (id_block,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_offset,
                  end_offset,
                  cg_port_num);
```

14.5.6.2 Parameter Definitions for Get Monthly Averages (Relative Times)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character array (declared as type CM50$DDT_string) that contains the ASCII name of the DDT to be used.

mpl_name—The name of a Multi-Point List structure (declared as type CM50_idb_rec) defining the data to be retrieved.

id_block—The name of a 16-byte variable (declared as type CM50$int8) containing the internal ID for an LCN tag. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of a shortword (declared as type CM50$int2) that specifies the maximum number of history items (1..262) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between `begin_offset` and `end_offset`, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with `status_table` entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times the number of points is greater than 598.

real_values_array—The name of a Real array (declared as type CM50$aver_array) where the history data is to be stored.

status_table—The name of a shortword array (declared as type CM50$hist_array) to contain the value status for each returned snapshot. If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX
CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an integer array (declared as type CM50$time_array) that will contain the time stamp in seconds for each returned average. See heading 15.3 for time-stamp conversions.

max_array—The name of a real array (declared as type CM50$aver_array) that will contain the maximum process value recorded in the averaged period. Due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of a real array (declared as type CM50$aver_array) that will contain the minimum process value recorded in the averaged period. Due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an array of integers (declared as type CM50$time_array) that will contain the number of samples used in calculating each returned average value.

begin_offset—The name of a shortword (declared as type CM50$int2) that indicates a relative offset from current LCN time that represents the first history record to be fetched.

end_offset—The name of a shortword (declared as type CM50$int2) that indicates a relative offset from the current LCN time representing the last history record to be fetched.

cg_port_num—The name of a shortword (declared as type CM50$int2 with a value of 1-4) identifying the CG to be accessed.

14.5.7 Get Monthly Averages (Absolute Times)

When a point is historized more often than once per minute, it is possible for the number of samples taken during a month to exceed the capacity of a 16-bit integer. This call provides a 32-bit integer count of the number of samples in a monthly average using absolute time.

NOTE

Retrieval of monthly averages using this call is only supported by LCN release 400 or later.

14.5.7.1 Example Pascal call for Get Monthly Averages (Absolute Times)

```
return_status := CM50_DDTHIS_MNTHT
                 (ddt_name,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time);
```

for Multi-Point Lists (instead of DDT):

```
return_status := CM50_MPLHIS_MNTHT
                 (mpl,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time,
                  cg_port_num);
```

for single point requests:

```
return_status := CM50_PTHIS_MNTHT
                 (id_block,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time,
                  cg_port_num);
```

14.5.7.2 Parameter Definitions for Get Monthly Averages (Absolute Times)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (declared as type CM50$DDT_string) that contains the ASCII name of the DDT to be used.

mpl —The name of an Multi-Point List structure (declared as type CM50_idb_rec) defining the data to be retrieved.

id_block—The name of a 16-byte variable (declared as type CM50$int8) containing the internal ID for an LCN tag. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of a shortword (declared as type CM50$int2) that specifies the maximum number of history items (1..261) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between begin and end times, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with status_table entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times (1 + the number of points) is greater than 598.

real_values_array—The name of a Real array (declared as type CM50$aver_array) where the history data is to be stored.

status_table—The name of a shortword array (declared as type CM50$hist_array) to contain the value status for each returned snapshot. If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX:
CM50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
CM50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
CM50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
CM50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
CM50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an integer array (declared as type CM50$time_array) to receive the time stamp in seconds for each returned average. See heading 15.3 for time-stamp conversions.

max_array—The name of a real array (declared as type CM50$aver_array) that will contain the maximum process value recorded in the averaged period. **Note:** Due to the data compression algorithm on the History module, there may be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of a real array (declared as type CM50$aver_array) that will contain the minimum process value recorded in the averaged period. **Note:** Due to the data compression algorithm on the History module, there may be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an array of integers (declared as type CM50$time_array) that will contain the number of samples used in calculating each returned average value.

begin_date_time—The name of a 14-character string (declared as CM50_lcn_asctim_type) in the format MM/DD/YYΔHH:MM (where Δ indicates a blank character) specifying the date and time for the most recent record to be fetched from the History Module.

The value entered for this parameter should be some time after the desired time period begins, but before the next time period's collection time. For example, begin_date_time for an 11:00 hourly average should be set to any time from 11:01 to 11:59.

end_date_time—The name of a 14-character string (declared as CM50_lcn_asctim_type) in the format MM/DD/YYΔHH:MM, specifying the date and time for the oldest record to be fetched from the History Module. The end_date_time must be earlier than the begin_date_time.

The value entered for this parameter should be some time after the previous period's collection, but before the desired time period begins. For example, to get an hourly average at 11:00, end_date_time should be set any time between 10:01 and 10:59.

cg_port_num—The name of a shortword (declared as type CM50$int2 with a value of 1-4) identifying the CG to be accessed.

14.5.8 Historization Sampling Rate Queries

These functions query the LCN and return the current Historization Sampling Rate (number of snapshots recorded each minute) for a point or set of points.

NOTE

Retrieval of sampling rates using this call is only supported by LCN release 400 or later.

14.5.8.1 Example Pascal calls for Query Sampling Rate

For Points referenced in a History DDT:

```
return_status := CM50_DDTHIS_RATE
                 (ddt_name,
                  history_rate_array,
                  status_table);
```

For a List of Internal Point ids:

```
return_status := CM50_MPLHIS_RATE
                 (mpl_name,
                  history_rate_array,
                  status_table,
                  cg_port_number);
```

For a Point addressed by its internal id:

```
return_status := CM50_PTHIS_RATE
                 (id_block,
                  history_rate,
                  cg_port_number);
```

For a Point addressed by its internal id:

```
return_status := CM50_TAGHIS_RATE
                 (tagname,
                  history_rate,
                  cg_port_number);
```

14.5.8.2 Parameter Definitions for History Sampling Rate Queries

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (CM50_HIS_PART), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (declared as type CM50$DDT_string) that contains the ASCII name of the DDT to be used.

mpl —The name of an Multi-Point List structure (declared as type CM50_idb_rec) defining the data to be retrieved.

id_block—The name of a 16-byte variable (declared as type CM50$int8) containing the internal ID for an LCN tag. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

tagname—The name of a 40-character string (declared as cm50_tagname_type) that contains an LCN tagname in the form "point.parameter(index)", where the "(index)" is used only to identify elements of an array.

history_rate —The name of a shortword (declared as CM50$int2) identifying the number of snapshots to collected each minute. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

history_rate_array —The name of an array of short words (declared as CM50$hist_array) identifying the number of snapshots collected each minute. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

status_table—The name of an array of short words (declared as CM50$hist_array) to contain the data access code for each point (See appendix A.1).

14.6 TEXT MESSAGE TRANSFERS

The two interface routines in this group are used to send and receive character-string messages over the LCN.

14.6.1 Get Message Interface

This routine is used to fetch a character-string message held in a buffer by this program's ACIDP. The message presence is determined as the result of a Get ACP Status request.

14.6.1.1 Example Pascal Call for Get Message

```
return_status := CM50_GETMSG
                 (msg,
                  msg_len);
```

14.6.1.2 Parameter Definitions for Get Message

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. There are three non-normal return_status values for this call that indicate the need for additional processing:

215000521	CM50_MSG_TRUNC	Received message was truncated
215000561	CM50_MSG_QUE	Message was received and another one is queued
215000571	CM50_MSG_QUET	Received message was truncated & another one is queued

msg—The name of a 120-character string (declared as type CM50$msg_string) where the message is to be stored.

msg_len—The name of a shortword (declared as type CM50$int2) that specifies the maximum number of characters to accept (120-character limit).

14.6.2 Send Message Interface

This routine is used to send a message to all operator stations assigned to the same unit as this program's ACIDP. A request to wait for operator confirmation is optional. If operator confirmation is requested, execution of the requesting program is suspended until either the confirmation occurs, or until its specified wait time expires. The requesting program receives an indication of whether confirmation or a time out occurs.

14.6.2.1 Example Pascal Call for Send Message

```
return_status := CM50_STOREMSG
                 (msg,
                  msg_len,
                  confirm,
                  timeout,
                  dest);
```

14.6.2.2 Parameter Definitions for Send Message

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

msg—The name of a character string (declared as type CM50$msg_string) that contains the message to be sent.

msg_len—The name of a shortword (declared as type CM50$int2) that specifies the number of characters to be transmitted. The maximum number of characters depends on message destination: 60 for CRT displays and 72 for printing. Over-length messages are truncated. All messages are archived if the HM is so configured.

confirm—The name of a Boolean shortword (declared as type CM50$bool2 where 1 = TRUE and 0 = FALSE) that specifies whether or not a message confirmation is required. Note that this parameter is treated as FALSE if the message destination is printer only.

timeout—The name of a shortword (declared as type CM50$int2) that specifies the number of seconds (0 to 3600) the system is to wait for confirmation before returning control to the requesting program with a "no confirm" return_status. (Allow for a built-in time lag of up-to-10 seconds.) The Wait Time parameter is ignored if the Confirm parameter is set to OFF or the message destination is printer only.

dest—The name of a shortword (declared as type CM50$int2) that specifies where the message is to be sent, as follows:

0 – CRT only
1 – Printer only
2 – Both

14.6.2.3 Event-Initiated Reports

Two types of Event-Initiated Reports can be invoked by specially formatted messages from an ACP or an Indirect Control Program to the Area Universal Stations:

- Logs, reports journals, and trends configured in the Area database
- Event History reports

Details of message requirements are given in Section 30 of the *Engineer's Reference Manual* located in the *Implementation/Startup & Reconfiguration* - 2 binder.

PROGRAM CONTROL AND SUPPORT (Pascal)
Section 15

This section discusses program interfaces that control the execution of ACPs and convert values between formats used in the VAX and on the TDC 3000 Local Control Network.

15.1 ACP EXECUTION SUPPORT

These interface routines affect the orderly execution and termination of application programs.

15.1.1 ACP Initialization Interface

This routine (or the vintage ACPTRP procedure) **must** be the first executable statement in each ACP but is optional for DAPs and Indirect Control Programs. It establishes a termination handler and ensures proper ACP table setup. Failure to invoke this interface routine as the first statement of an ACP may not appear to cause immediate problems, but will result in improper termination handling. The termination status is not reported to the CG, and the ACP appears to both the CM50S and the CG to still be in the RUN state even though the process has terminated.

The call to CM50_SET_ACP also establishes a system lock that allows the program to be terminated cleanly if CM50S is shut down. Therefore, it is advisable to include this call in every program that is mapped to the CM50S shareable image.

15.1.1.1 Example Pascal Call for ACP Initialization

```
return_status :=  CM50_SET_ACP  (reset);
```

15.1.1.2 Parameter Definitions for ACP Initialization

return_status—The name of an integer to receive the overall return status of the function. This function always returns as a success (`return_status` = 1).

reset—The name of an INTEGER*2 that specifies the reaction of the trap handler to an abort. If the ACP is aborted for any reason, the Abort code is recorded in the CG/PLNM database and the ACP Status table. If the value of **reset** is 1, then the execution status of the ACP is reset to OFF/DELAY regardless of how the program terminated. For any other value of **reset**, the execution status of the ACP becomes OFF/DELAY only after normal termination and is set to ABORT after an abnormal program termination.

15.1.2 Get ACP Status Interface

This routine fetches a set of parameters that enables the requesting ACP to determine why the system has turned it on and what special processing may be required at this time. It should be used during both the "setup" and "cleanup" program stages each time an ACP runs. After servicing this request, the interface routine resets its copy of these values in preparation for any subsequent ACP turn on.

NOTE

GETSTS is one of the few CM50S user-interface routines that is not implemented as a function. It is called as a Pascal subroutine.

15.1.2.1 Example Pascal Call for Get ACP Status

```
GETSTS (take_i_p,
        ps_msg,
        demand,
        procspec,
        scheduled,
        upper_level);
```

15.1.2.2 Parameter Definitions for Get ACP Status

take_i_p—The name of a Boolean shortword (declare as type CM50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE the first time this program is turned on by the CG, following an initialization event (see heading 4.4.1). `take_i_p` should be ignored when `upper_level` is TRUE.

ps_msg—The name of a Boolean shortword (declare as type CM50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE if a message for the program is waiting at the CG.

demand—The name of a Boolean shortword (declare as type CM50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on as the result of a process operator request.

procspec—The name of a Boolean shortword (declare as type CM50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on as the result of a process special to its ACIDP from an HG, AM, or another ACP.

scheduled—The name of a Boolean shortword (declare as type CM50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on by periodic or cyclic scheduling.

upper_level—The name of a Boolean shortword (declare as type CM50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on by the VAX.

15.1.3 ACP Delay Interface

This routine suspends execution of the calling program for a specified number of seconds. Program execution resumes at the statement following the delay call.

15.1.3.1 Example Pascal Call for ACP Delay

```
sleep :=    CM50_ACPDELAY
        (delay_time);
```

15.1.3.2 Parameter Definitions for ACP Delay

sleep—The name of a Boolean shortword (declare as type CM50$bool2 with 1 = TRUE and 0 = FALSE) that contains the overall return status of the function call. The value will be FALSE when the call has been rejected because of an invalid delay time value.

delay_time—The name of a shortword (declare as type CM50$int2) that contains the length of time (1 to 60 seconds) that the requesting program is to be suspended.

15.1.4 ACP Hibernate Interface

This routine suspends execution of the calling ACP (through a VMS SYS$HIBER request) until the next turn on request. The program and associated data remain in memory during hibernation, in effect making it memory-resident. Program execution resumes at the statement following the CM50_HIBER call.

15.1.4.1 Example Pascal Call for ACP Hibernate

```
hiber_stat :=    CM50_HIBER;
```

15.1.4.2 Parameter Definitions for ACP Hibernate

hiber_stat—The name of an integer to receive the overall return status of the function call. This value should always = 1 (SS$_NORMAL). Any other value indicates a fatal error. The program should call the GETSTS routine (see heading 15.1.2) to determine how the Wake call was issued.

15.1.5 ACP Termination Interface

This routine terminates the execution of the calling ACP. It must be used as the last operating statement of each ACP but is optional for DAPs and Indirect Control Programs.

For ACPs, this call stores a termination-status code in the associated ACIDP's ABORTCOD parameter. The termination code can be viewed at a Universal Station (see the definitions for ABORTCOD and EXECSTAT at heading 4.4.1), but in a revised form. The integer value assigned here is translated into two hexadecimal digits (00 to FF) and appended to the character string EA. Thus, an ACP-assigned abnormal termination code of 15 appears at the Universal Station display as EA0F.

If an ACP is aborted by the VMS operating system, an abort code of VMSF is stored in its ACIDP's ABORTCOD.

The execution state of an ACIDP can be changed from ABORT to normal by operator demand through a Universal Station or by invoking the ACP Operation screen's Deactivate/Terminate function. See heading 5.8 for abort recovery details.

NOTE

PRGTRM is one of the few user-interface routines that is not implemented as a function. It is called as a Pascal subroutine.

15.1.5.1 Example Pascal Call for ACP Termination

```
PRGTRM   (terminate_code);
```

15.1.5.2 Parameter Definitions for ACP Termination

terminate_code—The name of an integer that must contain zero or a positive value (1 to 255). Zero value indicates normal termination. Nonzero values are user-specified codes for nonnormal termination (abort). Note that if you provide a value outside the valid range, ABORTCOD will contain EAΔΔ (where.Δ represents a blank).

15.2 ENTITY NAME CONVERSIONS

The interface routines in this group convert ASCII references to tags on the LCN to their internal LCN identifiers.

NOTE

The all internal point.parameter addresses need to be rebuilt and the program(s) using them need to be recompiled whenever the LCN database is changed in a significant manner, such as by the rebuild or deletion of data points referenced in the address array.

15.2.1 Convert External to Internal ID

These routines fetch the internal ID of a point.parameter for the calling program. Use of the internal ID by repetitive single-value data gets and stores reduces system overhead and provides faster return of data. The specification of which point.parameter internal ID is wanted and where it is to be stored is contained in the call.

15.2.1.1 Example Pascal Calls for Convert ID

Using point and parameter names as separate variables:

```
return_status :=     CM50_CONV_PT
                     (entity,
                      param,
                      param_ix,
                      id_block,
                      val_typ,
                      cg_port_num);
```

When the external id is expressed as a Tag name (not separate point and parameter), use:

```
return_status :=     CM50_CONV_TAG
                     (tag_name,
                      id_block,
                      val_typ,
                      cg_port_num);
```

15.2.1.2 Parameter Definitions for Convert ID

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000146 (CM50_LCN_ARRAY)—the array size specified by `param_ix` is smaller the actual array size.

215000322 (CM50_ACC_SIZE)—the array size specified by `param_ix` is larger than the actual array size.

tag_name—The name of a 40-character string (declare as CM50_tag_name_type) that identifies the LCN value(s) to be stored. The tag name is formatted as "point.param (param_ix)".

entity—The name of a 20-character string (declare as CM50_Entity_name_type) that contains the ASCII Point ID. It should contain a point name of up to 16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter for Network Gateway routing.

param—An 8-character string that contains the ASCII name of the parameter to be converted.

param_ix—The name of a shortword (declare as type CM50$int2) specifying the parameter index. Use of this value is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, 5, 13, 15, 17 or 19, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of `param_ix` is used as an index and must be greater than zero. If the parameter being accessed is not an array type, `param_ix` must be zero.

When `val_typ` is 7, 8, 9, 10, 14, 16, 18 or 20, a whole array (or a subset of the array starting with the first element) is to be accessed and `param_ix` is used to specify the number of elements to be accessed. If `param_ix` is smaller than the actual array size, the conversion will be made; if it is larger than the actual array size, the conversion is not made. Both conditions cause non-normal `return_status` values to be returned.

id_block—The name of a 16-byte variable (declare as type CM50$int8) where the internal ID data block is to be returned. Save these eight values for later use in calls on this point.parameter. The ID data block contents are as follows:

Word 1—	Data type
Words 2..5—	Internal point identifier
Word 6—	Parameter subscript
Word 7—	Parameter qualifier (array size)
Word 8—	Enumeration set identifier

val_typ—The name of a shortword (declare as type CM50$int2) that contains a number that designates value type. If the incorrect value is supplied on input, this value will be updated as an output variable. The coded values:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
6 = not used
7 = Real array
8 = Integer array
9 = Enumeration array
10 = Ordinal value of enumeration array
13 = Internal entity id
14 = Internal entity id array
15 = External entity id
16 = External entity id array
17 = Time value
18 = Time value array
19 = String value
20 = String value array

cg_port_num—The name of a shortword (declare as type CM50$int2 with a value of 1-4) identifying the CG to be accessed.

15.2.2 Convert List of External IDs

These routines fetch an array of internal IDs for a list of point.parameters. These calls are designed for use with the Point Array calls described in section 14.2. All of the point.parameters in each list must be of the same data type (Real, ASCII, etc.).

15.2.2.1 Example Pascal Call for Convert Lists

When the point and parameter names are maintained separately, use:

```
return_status :=     CM50_CONV_PT_LIST
                     (entity_arr,
                      param_arr,
                      param_ix_arr,
                      number_of_values,
                      val_typ,
                      cg_port_num
                      id_block_arr,
                      return_arr);
```

When the external id is expressed as a Tag name (not separate point and parameter), use:

```
return_status :=     CM50_CONV_TAG_LIST
                     (tagname_arr,
                      number_of_values,
                      val_typ,
                      cg_port_num
                      id_block_arr,
                      return_arr);
```

15.2.2.2 Parameter Definitions for Convert Lists

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `return_arr` array entry for each returned id block must be checked for errors.

return_arr—The name of an array of up to 300 integers (declare as CM50_return_arr_type) to receive the status of the conversion of each point or tag. See Appendix A.2 for an explanation and a listing of all assigned return code values.

tagname_arr—The name of an array of up to 300 40-character strings (declare as CM50_tag_list_type). Each string contains the ASCII tagname of an LCN entity for which the internal ID is to be obtained. The tagnames are formatted as `Point.Parameter` or `Point.Parameter(ix)`, where (ix) is an array element index used only with array parameters.

entity_arr—The name of an array of up to 300 20-character strings (declare as CM50_entity_list_type), each containing an ASCII Point id. Each string should contain a point name of up to 16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter for Network Gateway routing.

param_arr—The name of an array of up to 300 8-character strings (declare as CM50_param_names_type), each containing the ASCII parameter name of a point.parameter for which the internal ID is to be obtained.

param_ix_arr—The name of an array of up to 300 shortwords (declare as type CM50_integer_array_type) used as array element index values corresponding to the individual parameter names in param_arr. For each non-array parameter named in that array, the corresponding value in this array should be zero.

cg_port_num—The name of a shortword (declare as type CM50$int2 with a value of 1-4) identifying the CG to be accessed.

id_block_arr—The name of an array of up to 300 16-byte variables (declare as CM50_point_list_array_type) where the internal ID data blocks are to be returned The ID data block contents are as follows:

Word 1—	Data type
Words 2..5—	Internal point identifier
Word 6—	Parameter subscript
Word 7—	Parameter qualifier (array size)
Word 8—	Enumeration set identifier

number_of_values—The name of a shortword (declare as type CM50$int2) that contains the number of points/tags in the list to be converted.

val_typ—The name of a shortword (declare as type CM50$Int2) that contains a number that designates LCN value type. This value must be supplied in the calling argument. The acceptable values are:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
13 = Internal entity id
15 = External entity id
17 = Internal Time
19 = String

15.3 VALUE CONVERSIONS

The interface routines in this group convert values between different formats used in the LCN and the host computer.

15.3.1 Valid Number Check

This routine checks a value of type "Real" to determine if it is a valid single-precision, floating-point number. Its primary purpose is to check for the "Bad Value" indicator, NaN (-0).

15.3.1.1 Example Pascal Call for Valid Number Check

```
value_st :=   CM50_VALIDN (value);
```

15.3.1.2 Parameter Definitions for Valid Number Check

value_st—The name of a Boolean shortword (declare as type cm50$bool2 where 1 = TRUE and 0 = FALSE) that returns TRUE if "Value" is found to be a valid floating-point number. It returns FALSE for minus zero (NaN) or other invalid bit configurations.

value—The name of a Real variable that contains a single-precision, floating-point value that is to be checked. When `value_st` returns FALSE, the contents of `value` have been changed to 0.0.

15.3.2 Set Bad Value

This routine stores the bad value constant, NaN (-0), into the specified Real variable.

15.3.2.1 Example Pascal Call for Set Bad Value

```
return_status := CM50_SETBAD (var_name);
                 or
CALL CM50_SETBAD (var_name)
```

15.3.2.2 Parameter Definitions for Set Bad Value

return_status—The name of an integer to receive the overall return status of the function call. For this function, `return_status` = 1 always.

var_name—The name of a Real variable where the bad value constant for the LCN is to be stored. Note that attempting to use a variable that has been set to Bad Value in an arithmetic or assignment statement will cause a fatal Pascal error at run time.

15.3.3 Convert Time Values

Within the CM50 environment, Date/time variables are often maintained in a variety of formats. The following routines convert time values from any one of the following formats to any other:

abbrev.	format	use
LCN	4-byte integer	internal LCN clock, number of seconds since January 1, 1979
VAXB	8-bytes (array of two 4-byte integers)	VAX binary system clock format
VAXA	22 characters	VAX standard ASCII time display: 'dd-MON-yyyy hh:mm:ss'
ASC	18 characters	LCN standard ASCII time display 'mm/dd/yy hh:mm:ss'
EURO	18 characters	European ASCII time display 'dd/mm/yy hh:mm:ss'
ARY	12 bytes (equivalenced to six 2-byte integers)	shortword array with element: 1 = year 2 = month 3 = day 4 = hour 5 = minute 6 = second

In each routine, the first argument must be assigned the input value and the second argument is the returned converted value.

15.3.3.1 Example Pascal Calls to Convert Time

Convert internal LCN time to a shortword array:

```
return_status :=  CM50_TIMLCN_ARY
                      (lcn,
                       ary);
```

Convert internal LCN time to an ASCII string:

```
return_status :=  CM50_TIMLCN_ASC
                      (lcn,
                       asc);
```

Convert internal LCN time to a European string:

```
return_status :=  CM50_TIMLCN_EURO
                      (lcn,
                       euro);
```

Convert internal LCN time to VAX display format:

```
return_status :=  CM50_TIMLCN_VAXA
                      (lcn,
                       vaxa);
```

Convert internal LCN time to VAX binary:

```
return_status :=  CM50_TIMLCN_VAXB
                      (lcn,
                       vaxb);
```

Convert a shortword array to internal LCN:

```
return_status :=  CM50_TIMARY_LCN
                      (ary,
                       lcn);
```

Convert a shortword array to an ASCII string:

```
return_status :=  CM50_TIMARY_ASC
                      (ary,
                       asc);
```

Convert a shortword array to a European string:

```
return_status :=  CM50_TIMARY_EURO
                      (ary,
                       euro);
```

Convert a shortword array to VAX display format:

```
return_status :=  CM50_TIMARY_VAXA
                      (ary,
                       vaxa);
```

Convert a shortword array to VAX binary:

```
return_status :=  CM50_TIMARY_VAXB
                      (ary,
                       vaxb);
```

Convert an ASCII string to internal LCN:

```
return_status :=  CM50_TIMASC_LCN
                      (asc,
                       lcn);
```

Convert an ASCII string to a shortword array:

```
return_status :=  CM50_TIMASC_ARY
                      (asc,
                       ary);
```

Convert an ASCII string to a European string:

```
return_status :=  CM50_TIMASC_EURO
                      (asc),
                       euro);
```

Convert an ASCII string to VAX display format:

```
return_status :=  CM50_TIMASC_VAXA
                      (asc,
                       vaxa);
```

Convert an ASCII string to VAX binary:

```
return_status :=  CM50_TIMASC_VAXB
                      (asc,
                       vaxb);
```

Convert a European string to internal LCN:

```
return_status :=  CM50_TIMEURO_LCN
                      (euro,
                       lcn);
```

Convert a European string to a shortword array:

```
return_status :=  CM50_TIMEURO_ARY
                      (euro,
                       ary);
```

Convert a European string to an ASCII string:

```
return_status :=  CM50_TIMEURO_ASC
                      (euro,
                       asc);
```

Convert a European string to a VAX display format:

```
return_status :=  CM50_TIMEURO_VAXA
                      (euro,
                       vaxa);
```

Convert a European string to VAX binary:

```
return_status :=  CM50_TIMEURO_VAXB
                      (euro,
                       vaxb);
```

Convert VAX display format to internal LCN:

```
return_status :=  CM50_TIMVAXA_LCN
                      (vaxa,
                       lcn);
```

Convert VAX display format to a shortword array:

```
return_status :=  CM50_TIMVAXA_ARY
                      (vaxa,
                       ary);
```

Convert VAX display format to an ASCII string:

```
return_status :=  CM50_TIMVAXA_ASC
                      (vaxa,
                       asc);
```

Convert VAX display format to a European string:

```
return_status = CM50_TIMVAXA_EURO
                      (vaxa,
                       euro);
```

Convert VAX display format to VAX binary:

```
return_status :=  CM50_TIMVAXA_VAXB
                      (vaxa,
                       vaxb);
```

Convert VAX binary to internal LCN:

```
return_status :=  CM50_TIMVAXB_LCN
                      (vaxb,
                       lcn);
```

Convert VAX binary to a shortword array:

```
return_status :=  CM50_TIMVAXB_ARY
                      (vaxb,
                       ary);
```

Convert VAX binary to an ASCII string:

```
return_status :=  CM50_TIMVAXB_ASC
                        (vaxb,
                         asc);
```

Convert VAX binary to a European string:

```
return_status :=  CM50_TIMVAXB_EURO
                        (vaxb,
                         euro);
```

Convert VAX binary to VAX display format:

```
return_status :=  CM50_TIMVAXB_VAXA
                        (vaxb,
                         vaxa);
```

15.3.3.2 Parameter Definitions for Convert Time Values

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

lcn—The name of an integer that contains a value representing internal LCN time to the nearest second.

ary—The name of a 12-byte string (declare as CM50_integer_time_type) that contains a value representing a date and time.

asc—The name of an 18-character string (declare as CM50$Time_arr) representing time in the format : 'mm/dd/yy hh:mm:ss '.

euro—The name of an 18-character string (declare as CM50$Time_arr) representing time in the format : 'dd/mm/yy hh:mm:ss '.

vaxa—The name of a 22-character string (declare as CM50_lcn_ASCII_time_type) representing time in the format: 'dd-MON-yyyy hh:mm:ss', where MON represents the first three letters (in upper case) of the English name of the month.

vaxb—The name of a 64-bit variable (declare as CM50_VMS_Binary_time_type) that contains a value representing internal VAX binary time.

CM50S ADMINISTRATION (Pascal)
Section 16

This section discusses the programmatic calls that can be used to manage the ACPs and DDTs installed in a CM50S system.

16.1 PROGRAMMATIC INTERFACES TO ACP OPERATIONS

A programmatic interface to all ACP Operations gives users programmatic access to the same ACP functions that are available through the ACP Operations user interface. In order to use the ACP Programmatic Interface, the user should include the ACP Include files (`CM50_FLAGS_INCLUDE.PAS` and `CM50_ACP_INCLUDE.PAS`). These files define data types and routines required by the Programmatic Interface calls. The following sections discuss the ACP Programmatic Interface calls in detail.

16.1.1 Install ACP

This routine is called to install an ACP. The ACP can be installed under a different name than the executable filename.

16.1.1.1 Example Pascal Call for Install ACP

```
return_status := CM50_ACP_INSTALL
                 (acp_name,
                  process_name,
                  mailbox_name,
                  exe_path,
                  mode,
                  input_path,
                  output_path,
                  error_path,
                  privilege,
                  uic,
                  priority,
                  creprc_flags,
                  quota_list,
                  flags);
```

16.1.1.2 Parameter Definitions for Install ACP

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name of the ACP (packed array of 12 characters, declare as ACP_NAME_TYPE) to be installed. Must be specified.

process_name—The Name (packed array of 15 characters, declare as PROC_NAME_TYPE) to be assigned to the created process. If set to spaces, the ACP name will be used. Note: Each process must have a unique name. The activation of an ACP will fail if a process with the specified process_name is active on the system.

mailbox_name—A VMS termination mailbox name (packed array of 40 characters, declare as MBX_LOG_TYPE --normally set to spaces) to receive a termination message when the created process (ACP) is complete. This is a temporary termination mailbox created by the Programmatic Interface and ACPOPER utility. For more information, refer to the VMS System Services Reference Manual. This mailbox parameter is applicable only when the ACP is executed as a remote (detached) process. An ACP run interactively ignores the mailbox parameter in the ACP table. The mailbox is created using VMS defaults.

exe_path—Full pathname (packed array of 80 characters, declare as PATH_NAME_TYPE) of the executable file. If set to spaces, the default is the executable file specified by the acp_name in the CM50$ACP directory.

mode— 16-bit integer (declare as ACP_MODE_TYPE) Specifies what mode to install the ACP in. The values (and their symbolic names) are:

1 = TEST
2 = RESTRICTED
3 = NORMAL

input_path—Pathname (packed array of 80 characters, declare as PATH_NAME_TYPE) of the alternate input filename.

output_path—Pathname (packed array of 80 characters, declare as PATH_NAME_TYPE) of the alternate output filename. If left blank, SYS$OUTPUT will be directed to the NULL device.

error_path—Pathname (packed array of 80 characters, declare as PATH_NAME_TYPE) of the alternate error filename. If left blank, SYS$ERROR will be directed to the NULL device.

privilege—Privileges specification. Declared as PRIV_MASK_TYPE assigns special VMS privileges to the ACP. Set both components (.L0 and .L1) to zero for a normal, unprivileged ACP.

uic—Name of user (packed array of 12 characters, declare as UIC_TYPE) whose UIC is to be used when the ACP is executed remotely. Only the first 12 characters are significant, the remainder should be blank filled.

priority—32-bit integer (declare as PRIORITY_TYPE) specifying the VMS priority (0-30) of the ACP process.

creprc_flags—32-bit integer (declare as CREPRC_FLAG_TYPE) VMS Create Process flags specification. Normally set to zero.

quota_list—Quotas specification. Declared as QUOTA_LIST_TYPE (an array of 14 QUOTA_TYPE records). The last element of the array must have a QUOTA_TAG = zero. To use the system defaults, pass a single element with a value of zero.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_ACIDP_ACTIVATE  (required to activate an ACP which is
                        connected to an ACIDP)
CM50$M_HANDLER
CM50$M_MSGON
```

16.1.2 Uninstall ACP

This routine is called to uninstall an ACP.

16.1.2.1 Example Pascal Call for Uninstall ACP

```
return_status := CM50_ACP_UNINST
                 (acp_name,
                  flags);
```

16.1.2.2 Parameter Definitions for Uninstall ACP

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name of the ACP (packed array of 12 characters, declare as ACP_NAME_TYPE) that is to be uninstalled.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.1.3 Activate ACP

This routine is called to activate an installed ACP under a mode specified by the user.

16.1.3.1 Example Pascal Call for Activate ACP

```
return_status := CM50_ACP_ACT
                 (acp_name,
                  mode,
                  flags);
```

16.1.3.2 Parameter Definitions for Activate ACP

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name of the ACP (packed array of 12 characters, declare as ACP_NAME_TYPE) to be activated.

mode—16-bit integer (declare a ACT_MODE_TYPE) that specifies whether the ACP is to be activated as a REMOTE detached process (mode := 0) or as an INTERACTIVE subprocess (mode := 1).

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.1.4 Deactivate ACP

This routine is called to deactivate an installed ACP, placing it in a specified state.

16.1.4.1 Example Pascal Call for Deactivate ACP

```
return_status := CM50_ACP_DEACTIVATE
                 (acp_name,
                  state,
                  flags);
```

16.1.4.2 Parameter Definitions for Deactivate ACP

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name of the running ACP (packed array of 12 characters, declare as ACP_NAME_TYPE) to be deactivated.

state—16-bit integer (declare as EXEC_STATE_TYPE) that specifies whether to set the ACIDP to an ABORT (state := 0) or OFF (state := 3).

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.1.5 Connect ACP to an ACIDP

This routine is called to connect an installed ACP to an ACIDP on the LCN.

16.1.5.1 Example Pascal Call for Connect ACP to an ACIDP

```
return_status := CM50_ACP_CONNECT
                 (acp_name,
                  acidp_name,
                  cg_port_number,
                  flags);
```

16.1.5.2 Parameter Definitions for Connect ACP to an ACIDP

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name of the ACP (packed array of 12 characters, declare as ACP_NAME_TYPE) to be connected.

acidp_name—The name of the ACIDP (packed array of 16 characters, declare as CM50_LONG_ACIDP) to connect the to the ACP.

cg_port_number—16-bit integer (declare as CG_NUM_TYPE) that specifies which CG (1-4) contains the `ACIDP.`

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.1.6 Disconnect ACP from an ACIDP

This routine is called to disconnect an installed ACP from an ACIDP on the LCN.

16.1.6.1 Example Pascal Call for Disconnect ACP from an ACIDP

```
return_status := CM50_ACP_DISCON
                 (acp_name,
                  flags);
```

16.1.6.2 Parameter Definitions for Disconnect ACP from an ACIDP

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name of the ACP (packed array of 12 characters, declare as ACP_NAME_TYPE) to be disconnected.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.1.7 Change ACP Mode

This routine is called to change the installation mode of an ACP.

16.1.7.1 Example Pascal Call for Change ACP Mode

```
return_status := CM50_ACP_CHG_MODE
                 (acp_name,
                  mode,
                  flags);
```

16.1.7.2 Parameter Definitions for Change ACP Mode

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name of the ACP (packed array of 12 characters, declare as ACP_NAME_TYPE) whose mode is to be changed.

mode—16-bit integer (declare as ACP_MODE_TYPE) that specifies the new mode of the ACP. Permitted values (and their symbolic names) are:

1 = TEST
2 = RESTRICTED
3 = NORMAL

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

`CM50$M_HANDLER`
`CM50$M_MSGON`

16.1.8 Get ACP Summary

This routine is called to get summary information for an installed ACP. The output optionally can be sent to the printer.

16.1.8.1 Example Pascal Call for Get ACP Summary

```
return_status := CM50_ACP_SUM
                 (acp_name,
                  acp_summary,
                  flags);
```

16.1.8.2 Parameter Definitions for Get ACP Summary

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name of the ACP (packed array of 12 characters, declare as ACP_NAME_TYPE) for which summary information is to be returned.

acp_summary—This parameter specifies where the summary information is to be returned. Declared as an ACP_SUMMARY_REC; the specific contents are described in the CM50_ACP_INCLUDE file.

Note that the ACP_SUMMARY_REC record can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.1.9 Get List of ACPs

This routine is called to get a list of installed ACPs. Up to 400 ACPs will be reported in a single call, if more than 400 ACPs are installed on a system, repeating the call with a start_rec of 1, 401 and 801 will return additional ACPs until the system maximum of 1000 have been returned.

16.1.9.1 Example Pascal Call for Get List of ACPs

```
RETURN_STATUS := CM50_ACP_LISTALL
                 (start_rec,
                  end_rec,
                  total_returned,
                  list,
                  flags);
```

16.1.9.2 Parameter Definitions for Get List of ACPs

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

start_rec—An Integer specifying the starting (lowest) record number within the ACP status table to be reported.

end_rec—An Integer specifying the ending (highest)record number within the ACP status table to be reported.

total_returned—An integer value specifying the number of records actually returned. This may be less than the number requested if the end of the table was reached.

list—An array 400 of ACP_SUMMARY_REC records (declare as ACP_LIST_REC type) receives the data requested. It must be dimensioned large enough to for the number of records requested (1 + end_rec - start_rec).

Note that the ACP_SUMMARY_REC record can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.2 PROGRAMMATIC INTERFACE TO DDT OPERATIONS

All CM50S DDT operations except for `Edit` can be accessed through the Programmatic Interface. All Programmatic Interface routines are called as functions, and the status of each call is returned as the value of the function call. The calling program must include the CM50_FLAGS_INCLUDE file and the CM50_DDT_INCLUDE file. Both files must be appropriate to the language being used (see heading 2.9.1).

Exception handling is provided by standard VMS condition handling routines or by custom routines written by the user. The Programmatic calls for all DDT functions are described in detail in the following paragraphs.

16.2.1 Build/Rebuild DDT

This routine is called to build, or rebuild, a DDT binary file from a DDT source file. Flag options include CG residence for the DDT and DDT Rebuild.

16.2.1.1 Example Pascal Call for Build/Rebuild DDT

```
return_status := CM50_DDT_BUILD
                 (ddt_name,
                  source_path,
                  cg_port_number,
                  description,
                  flags);
```

16.2.1.2 Parameter Definitions for Build/Rebuild DDT

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The name of the DDT (packed array of 9 characters, declare as DDT_NAME_TYPE) to be used to retrieve data. It may be left blank if the full source path is specified.

source_path—Pathname of the DDT source file (packed array of 80 characters, declare as PATH_NAME_TYPE). If set to spaces, the default is the DDT name in the current directory.

cg_port_number—16-bit integer (declare as CG_NUM_TYPE) which specifies which CG the DDT is associated with.

description—A text description (packed array of 36 characters, declare as DDT_DESC_TYPE) of the DDT being built. If the DDT source file specifies a Description, that description will be used and the value of this argument is ignored.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_REBUILD_DDT
CM50$M_DMP_DDT_ERRORS
CM50$M_MSGON
CM50$M_NO_SOURCE_DEBUG
CM50$M_CG_RES
CM50$M_WRITE_VT
```

NOTE

If the DDT (or another DDT by the same name) has already been built, then the CM50$M_REBUILD_DDT must be set ON.

For a new DDT, the CM50$M_REBUILD_DDT flag must be OFF.

16.2.2 Delete DDT

This routine is called to delete a DDT that already exists in the DDT table. If the DDT is installed in the CG, it is removed.

16.2.2.1 Example Pascal Call for Delete DDT

```
return_status := CM50_DDT_DELETE
                 (ddt_name,
                  flags);
```

16.2.2.2 Parameter Definitions for Delete DDT

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The name of the DDT (packed array of 9 characters, declare as DDT_NAME_TYPE) to be deleted.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.2.3 Get DDT Summary

This routine is called to summarize the specifications of a particular DDT.

16.2.3.1 Example Pascal Call for Get DDT Summary

```
return_status := CM50_DDT_SUM
                 (ddt_name,
                  summary,
                  flags);
```

16.2.3.2 Parameter Definitions for Get DDT Summary

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The name of the DDT (packed array of 9 characters, declare as DDT_NAME_TYPE) that is to be summarized.

summary—This argument (declare as type DDT_SUMMARY_REC) receives the requested information. Its contents are:

- Name of the DDT being summarized
- Pathname of the DDT's source file
- Description of the DDT
- Date that the DDT was first built
- Name of the original builder
- Most recent time the DDT was modified
- Installation status of the DDT
- Number of points in the DDT
- DDT Type—Input, Generic Input, Output, Generic Output, or History
- CG number that the DDT is associated with
- Whether or not DDT is installed in CG
- Name of ACIDP DDT is connected to
- Prefetch triggers (A packed array of 8 Booleans where:
  prefetch_trig[8] is the SCHeduled prefetch,
  prefetch_trig[7] is the PPS prefetch, and
  prefetch_trig[6} is the DMD (demand) prefetch.

Note that the DDT_SUMMARY_REC record can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.2.4 Get List of DDTs

This routine is called to retrieve a list of DDT summaries. Up to 500 DDT summaries may be returned in a single call. If more DDTs are installed on your system, they may all be retrieved by repeating this all with start_record set to 1, 501, 1001 and 1501 on successive calls.

16.2.4.1 Example Pascal Call for Get List of DDTs

```
return_status := CM50_DDT_LIST
                 (start_record,
                  end_record,
                  count,
                  list,
                  flags);
```

16.2.4.2 Parameter Definitions for Get List of DDTs

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

start_record—This shortword (declare as type INTV) specifies the number of the first DDT to retrieve.

end_record—This shortword (declare as type INTV) specifies the number of the last DDT to retrieve.

count—This shortword (declare as type INTV) receives the actual number of DDT records returned to the caller.

list—This argument will receive an array of DDTs information. This array of 500 DDT_SUMMARY_RECs is declared as DDT_ITEMS_ARR. The information returned for each record is

- Name of the DDT being summarized
- Pathname of the DDT's source file
- Description of the DDT
- Date that the DDT was first built
- Name of the original builder
- Most recent time the DDT was modified
- Installation status of the DDT
- Number of points in the DDT
- DDT Type—Input, Generic Input, Output, Generic Output, or History
- CG number that the DDT is associated with
- Tells whether DDT is installed in CG
- Name of ACIDP DDT is connected to

Note that the DDT_SUMMARY_REC record can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.2.5 Get DDT Detail

This routine is called to retrieve the detail information for the named DDT.

16.2.5.1 Example Pascal Call for Get DDT Detail

```
return_status := CM50_DDT_DETAIL
            (ddt_name,
             summary,
             data,
             points,
             details,
             values,
             flags);
```

16.2.5.2 Parameter Definitions for Get DDT Detail

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The name of the DDT (packed array of 9 characters, declare as DDT_NAME_TYPE) being summarized.

summary—A record (declare as type DDT_SUMMARY_REC) containing:

`DDT_Name`	Name of the DDT being summarized
`DDT_Sourc_Loc`	Pathname to the DDT's source file
`DDT_Desc`	Description of the DDT
`Built_On`	Date that the DDT was first built
`Built_By`	Tells who the original builder was
`Modified_On`	Most recent time the DDT was modified
`DDT_Status`	Installation status of the DDT
`Number_of_Pts`	Number of points in the DDT
`DDT_Type`	Input, Output, or History DDT
`CG_Port_Num`	CG that the DDT is associated with
`In_CG`	Tells whether DDT is installed in CG

Note that the DDT_SUMMARY_REC record can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

data—A record (declare as DDT_DATA_TYPE) containing:

`DDT_Types`	Names data types found in the DDT
`TTL_Each_Type`	Counts for each data type found

points—An array of 300 point name records (declare as type POINTS_ARR) containing:

`Point_Name`	Point name
`Param_Name`	Parameter name (with index)

Note that the POINTS_ARR record can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

details—An array of 300 records (declare as type DETAIL_REC_ARR) containing:

`Process_Type`	Real, Integer, ASCII, Enumeration, or Ordinal
`Dest_Src`	Destination or Source offset value
`Test`	Use test Y/N and test data value
`BVS`	Bad value substitution Y/N and data
`Algo`	Algorithm number selection and data
`Limits`	Limit check Y/N and data

values—An array of 300 records (declare as type VALUES_ARR). This argument will contain useful information only if full Table Processing (including a Values Table) is being used with the DDT. It contains the values from the last use of the DDT showing the values before and after table processing conversions. Any LCN Real data Bad Values are returned as zeros.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.2.6 Connect DDT to ACIDP

This routine is called to connect a DDT to an ACIDP for the purpose of enabling the Data Prefetch Function in the CG. The ACIDP-ACP connection must already exist and the DDT must be CG-resident and not already connected to an ACIDP.

The `ddt_name`, and either the `acp_name`, or `acidp_name` parameters are required in the call. The Schedule, PPS and Demand parameters also are required.

16.2.6.1 Example Pascal Call for Connect DDT to ACIDP

```
return_status := CM50_DDT_CONNECT
                 (ddt_name,
                  acidp_name,
                  acp_name,
                  trigger,
                  flags);
```

16.2.6.2 Parameter Definitions for Connect DDT to ACIDP

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The name of the DDT (packed array of 9 characters, declare as DDT_NAME_TYPE) that is to be connected to an ACIDP.

acidp_name—The name of the ACIDP (packed array of 16 characters, declare as CM50_LONG_ACIDP) to which the DDT is to be connected. The `acidp_name` can be blanks if a valid `acp_name` is provided.

acp_name—The name of the ACP (packed array of 12 characters, declare as ACP_NAME_TYPE) connected to the ACIDP to which the DDT is to be connected. The `acp_name` can be blanks if a valid `acidp_name` is provided

trigger—A code (packed array of 8 Boolean values, declare as DDT_TRIGGER_TYPE) with the three high-order bits assigned these meanings:

- Element 1 (bit 7): `Schedule`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Element 2 (bit 6): `PPS` (Point_Process_Special)—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Element 3 (bit 5): `Demand`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.2.7 Disconnect DDT from ACIDP

This routine is called to disconnect a DDT from an ACIDP. At least one of the three parameters, `ddt_name`, `acp_name`, or `acidp_name` is required in the call (the others are passed as blanks). The ACIDP-ACP-DDT connection must already exist.

16.2.7.1 Example Pascal Call for Disconnect DDT from ACIDP

```
return_status := CM50_DDT_DISCONNECT
                 (ddt_name,
                  acidp_name,
                  acp_name,
                  flags);
```

16.2.7.2 Parameter Definitions for Disconnect DDT from ACIDP

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The name of the DDT (packed array of 9 characters, declare as DDT_NAME_TYPE) that is to be disconnected. Can be blanks if either `acidp_name` or `acp_name` contains a valid name.

acidp_name—The name of the ACIDP (packed array of 16 characters, declare as CM50_LONG_ACIDP) from which the DDT is to be disconnected. Can be blanks if either `ddt_name` or `acp_name` contains a valid name.

acp_name—The name of the ACP (packed array of 12 characters, declare as ACP_NAME_TYPE) connected to the ACIDP from which the DDT is to be disconnected. Can be blanks if either `ddt_name` or `acidp_name` contains a valid name.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.2.8 Modify Triggers

This routine is called to modify the Triggers associated with a DDT that is connected to an ACIDP. At least one of the three parameters, `ddt_name`, `acp_name`, or `acidp_name`, is required in the call (the others are passed as blanks). The ACIDP-ACP-DDT connection must already exist.

16.2.8.1 Example Pascal Call for Modify Triggers

```
return_status := CM50_DDT_TRIGGERS
                 (ddt_name,
                  acidp_name,
                  acp_name,
                  trigger,
                  flags);
```

16.2.8.2 Parameter Definitions for Modify Triggers

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The name of the DDT (packed array of 9 characters, declare as DDT_NAME_TYPE) that is connected to the specified ACIDP. Can be blanks if either `acidp_name` or `acp_name` contains a valid name.

acidp_name—The name of the ACIDP (packed array of 16 characters, declare as CM50_LONG_ACIDP) to which the specified DDT is connected. Can be blanks if either `ddt_name` or `acp_name` contains a valid name.

acp_name—The name of the ACP (packed array of 12 characters, declare as ACP_NAME_TYPE) connected to the specified ACIDP. Can be blanks if either `ddt_name` or `acidp_name` contains a valid name.

trigger—A code (packed array of 8 Boolean values, declare as DDT_TRIGGER_TYPE) with the three high-order bits assigned these meanings:

- Element 1 (bit 7): `Schedule`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Element 2 (bit 6): `PPS` (Point_Process_Special)—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Element 3 (bit 5): `Demand`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.2.9 Install DDT Into CG

This routine is called to install the DDT into the CG.

16.2.9.1 Example Pascal Call for Install DDT Into CG

```
return_status := CM50_DDT_INSTALL
                 (ddt_name,
                  flags);
```

16.2.9.2 Parameter Definitions for Install DDT Into CG

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The name of the DDT (packed array of 9 characters, declare as DDT_NAME_TYPE) to be installed into the CG.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.2.10 Uninstall DDT from CG

This routine is called to remove a DDT from the CG.

16.2.10.1 Example Pascal Call for Uninstall DDT from CG

```
return_status := CM50_DDT_UNINST
                 (ddt_name,
                  flags);
```

16.2.10.2 Parameter Definitions for Uninstall DDT from CG

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The name of the DDT (packed array of 9 characters, declare as DDT_NAME_TYPE) to be removed from the CG.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The following flags apply to this call:

```
CM50$M_HANDLER
CM50$M_MSGON
```

16.3 PROGRAMMATIC INTERFACE TO CG DATABASE

These functions return information about the current points configured in the database of any CG connected to the CM50. The language specific declarations for these functions are contained in `CM50_CGDATA_INCLUDE.pas`

16.3.1 Resident DDT Summary

This function returns a list of all of the DDTs currently resident in the CG.

16.3.1.1 Example Pascal Call for Resident DDT List

```
return_status := CM50_CG_RDDT
                 (cg_port_num,
                  number_of_values,
                  ddt_list);
```

16.3.1.2 Parameter Definitions for Resident DDT List

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of an 16-bit integer (declare as CM50_CG_BASE_TYPE) identifying the CG to be accessed.

number_of_values—The name of an Integer that returns the number of DDTs currently installed as resident in the CG.

ddt_list—The name of an array of 40 DDT names (packed array of 10 characters, declare as type CM50_ResDDT_LIST) that will receive the list.

16.3.2 Calculated Results Data Points List

This function returns a list of all of the CRDPs currently configured in the CG.

16.3.2.1 Example Pascal Call for CRDP List

```
return_status := CM50_CG_CRDP
                 (cg_port_num,
                  number_of_values,
                  crdp_list);
```

16.3.2.2 Parameter Definitions for CRDP List

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of an 16-bit integer (declare as CM50_CG_BASE_TYPE) identifying the CG to be accessed.

number_of_values—The name of an Integer that returns the number of CRDPs currently configured in the CG.

crdp_list—The name of an array of 500 variables (packed arrays of 8 characters, declare as type CM50_CRDP_LIST) that will contain the names of the CRDPs.

Note that the CM50_CRDP_LIST structure can vary for different releases of CM50S, soprograms using this call should be recompiled when CM50S is upgraded.

16.3.3 ACIDP Detail

This function returns information about the current status of a specific ACIDP.

16.3.3.1 Example Pascal Call for ACIDP Detail

```
return_status := CM50_CG_ADETAIL
                 (cg_port_num,
                  acidp_record);
```

16.3.3.2 Parameter Definitions for ACIDP Detail

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of an 16-bit integer (declare as CM50_CG_BASE_TYPE) identifying the CG to be accessed.

number_of_values—The name of an Integer that returns the number of ACIDPs currently configured in the CG.

acidp_record—The name of a record (declare as type CM50_ACIDP_REC) with the following format:

ACIDP	: 8-character name of the ACIDP
DESC	: 24-character descriptor of the ACIDP
UNIT	: 2-character LCN Unit to which the ACIDP is assigned
KEYWORD	: 8-character LCN alias for the ACIDP
ACP	: 12-character name of the connected ACP
MODE	: 8-character enumerated value of the Program Mode
EXEC	: 8-character enumerated value of the Execution State
ACCES	: 8-character enumerated value of the Data Access Mode
DDT	: 9-character name of attached DDT
ACTYP	: 8-character enumerated value of the Activation Type
INHIB	: 8-character enumerated value of the Inhibit flag
STIME	: 8-character value of the Scheduled Start Time
PERIOD	: 8-character value of the Schedule Cycle Period
NXTTIM	: 18-character value of the Next Scheduled Activation Time
TAKEIP	: 4-character enumerated value of the Take_Initial_Path flag
RUNINIT	: 4-character enumerated value of the Run_on_Initialization flag
CONFWT	: 4-character enumerated value of the Confirm_Wait flag
CONFRQ	: 4-character enumerated value of the Confirm_Request flag
SCH	: 4-character enumerated value of the Schedule Activation flag
PPS	: 4-character enumerated value of Program_Special Activation flag
DMD	: 4-character enumerated value of Operator_Demand Activation flag
GROUP	: unsigned 16-bit integer (type INTV) value of the Group code.

Note that the CM50_ACIDP_REC structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

16.3.4 ACIDP Summary

This function returns a list of all of the ACIDPs configured in the CG.

16.3.4.1 Example Pascal Call for ACIDP Summary

```
return_status := CM50_CG_ACIDP
                 (cg_port_num,
                  number_of_values,
                  acidp_list,
                  acp_list,
                  mode_list,
                  state_list,
                  ddt_list,
                  trigger_list);
```

16.3.4.2 Parameter Definitions for ACIDP Summary

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of an 16-bit integer (declare as CM50_CG_BASE_TYPE) identifying the CG to be accessed.

number_of_values—The name of an Integer that returns the number of ACIDPs currently configured in the CG.

acidp_list—The name of an array of 250 variables (packed arrays of 8 characters, declare as type CM50_ACIDP_LIST) that will hold the names of the resident ACIDPs.

The CM50_ACIDP_LIST structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

acp_list—The name of an array of 250 variables (packed arrays of 12 characters, declare as type CM50_cgACP_LIST) that will contain the names of the ACPs connected to the corresponding ACIDP.

mode_list—The name of an array of 250 variables (16-bit integers declare as typeCM50_cgENUM_LIST) that will contain the numeric code for the installation mode of each ACIDP.

state_list—The name of an array of 250 variables (16-bit integers declare as typeCM50_cgENUM_LIST) that will contain the numeric code for the current execution state of each ACIDP.

ddt_list—The name of an array of 250 variables (packed arrays of 10 characters, declare as type CM50_cgDDT_LIST) that will contain the names of the DDT (if any) connected to the corresponding ACIDP.

trigger_list—The name of an array of 250 Trigger records (declare as type CM50_TRIG_LIST),where each Trigger record is an array of 3 Boolean 16-bit integers (indicating by 1 or 0 whether or not the connected DDT will be prefetched when the ACIDP is triggered on Schedule, Operator_Demand, or PPS).

16.3.5 LCN Configuration

This function returns information about the LCN configuration parameters for a specified CG. Note that the specified CG must be running TDC 3000 release 400 or later for this function to get a successful return_status.

16.3.5.1 Example Pascal Call for LCN Configuration

```
return_status := CM50_CG_CONFIG
                 (cg_prot_num,
                  cgconfig_record) ;
```

16.3.5.2 Parameter Definitions for LCN Configuration

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, return_status = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_prot_number—The name of a 16-bit integer (declare as CM50_CG_BASE_TYPE) identifying the CG (1 to 4) to be accessed.

cgconfig_record—The name of a record (declare as type CM50_CGCONFIG_REC) with the following fields:

LCN_VER	:	16-bit integer (type INVT) TDC 3000 software release level
LCN_REV	:	16-bit integer (type INVT) TDC 3000 revision level (future)
LCN_NODE	:	16-bit integer (type INVT) LCN node number of this CG
CG_VER	:	16-bit integer (type INVT) CG personality software release
CG_REV	:	16-bit integer (type INVT) CG personality software revision
TIME_SYNCH	:	16-bit integer (type INVT) CG Time synchronization period
CONFIRM_TIME	:	16-bit integer (type INVT) CG Time out for message confirm
CG_STATION	:	16-bit integer (type INVT) HDLC station number of the LCN
T1_TIME	:	16-bit integer (type INVT) T1 timer
N2_COUNT	:	16-bit integer (type INVT) Retry count
FLOAT_FORMAT	:	16-bit integer (type INVT) Floating point format (should be 2 for IEEE)
BAUD_RATE	:	16-bit integer (type INVT) enumeration (0= 1200, 1= 1760, 3= 2400, 5= 4800, 7= 9600, 8= 19200, 13= 38400, 14= 56700, 15= 76800)
TAG_SIZE	:	16-bit integer (type INVT) (0 for 8-character maximum, 1 for 16-characters)
HM_USER_MIN	:	16-bit integer (type INVT) number of minutes in a user average
HM_SHIFT_WK	:	16-bit integer (type INVT) number of shifts per week
HM_START_HR	:	16-bit integer (type INVT) daily/weekly averages starting hour (0 starts Sunday morning just after midnight)
HM_MONTH_TYP	:	16-bit integer (type INVT) (0 is calendar, 1 is 28-day cycles)
PINID	:	2-character identifier of this LCN for Network Gateway routing
DESCR	:	40-character CG descriptor on the LCN

16.4 PROGRAMMATIC INTERFACE TO FILE TRANSFER

These functions execute LCN file transfer commands programmatically. The calling program must include the CM50_FLAGS_INCLUDE.PAS and CM50_FTF_INCLUDE.PAS files in its source to insure that the functions and arguments are properly declared.

The DATAOUT facility allows the user, when requesting the execution of specific file transfer transactions, to place relevant data in the dataout or catalog file. This dataout file is a shared file by all concurrent users of file transfer. For example, user "Jones" requests a CM50_FILE_CATALOG transaction, the results of which are placed into the current dataout file. User "Smith" then requests a CM50_VOLUME_CATALOG transaction. These results also are placed into the same (current) dataout file.

CM50_FILE_CATALOG and CM50_VOLUME_CATALOG are the only file transfer transactions that require a dataout file. Other file transfer transactions treat dataout as an option for journalizing activity.

16.4.1 Read LCN File

This procedure will transfer a single file from an LCN NET volume to CM50S. Wildcard transfers of files are not supported. This procedure will also create an "LCN ATTRIBUTES" file for every LCN file that is transferred. Multiple copies of the same file, within the same VMS directory, are not allowed. The version number of the attributes file should remain 1. For more information regarding file attributes refer to the WRITE file procedure.

16.4.1.1 Example Pascal Call for File Read

```
Return_status  :=  CM50_LCN_READ
                     (lcn_file,
                      host_file,
                      acidp_name,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.1.2 Parameter Definitions for File Read

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx.

host_file—VMS pathname (packed array of 80 characters, declare as PATH_NAME_TYPE) to be used to store the LCN file (and its associated attributes file). If no extension is specified, the VMS default of .DAT will be used. If no directory is specified, the user's current default directory will be used. The LCN attributes file will use the following naming convention: the filename suffix or extension will be preceded by an under-bar character, followed by a period "LA" extension. For example; the LCN filename of FORMULAE.CL would have an attribute file of FORMULAE_CL.LA. Note: The transfer will fail if the pathname matches that of an existing file.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.2 Write LCN File

This procedure will transfer a single file from CM50S to LCN NET volume. This procedure requires the LCN ATTRIBUTES file for every LCN file that is transferred. Multiple copies of an LCN FILE within the same VMS directory, are allowed. These files would have been created by modifying the original LCN FILE which was transferred as version 1. The version number of the attributes file should be 1.

16.4.2.1 Example Pascal Call for File Write

```
Return_status  :=  CM50_LCN_WRITE
                     (host_file,
                      lcn_file,
                      acidp_name,
                      file_code,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.2.2 Parameter Definitions for File Write

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

host_file—VMS pathname (packed array of 80 characters, declare as PATH_NAME_TYPE) of the file to be transferred to the LCN If no directory is specified, the user's current default directory will be used. The associated LCN attributes file (with an extension of .LA) must be in the same directory.

lcn_file—LCN pathname (packed array of 28 characters declare as type LCN_PATH_NAME) where the file is to be stored on the LCN. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

file_code—Name of a shortword (declare as WRITE_FILE_TYPE) that determines whether the LCN file is to be replaced if it already exists at the LCN NET volume. The default is to abort the write if the file already exists. The enumerated values are:

`replace_write` = 0: Replace existing file
`abort_write` = 1: Return an error if the file already exists.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.3 List LCN File Attributes

This request will retrieve the file attributes for a specific LCN file. Wildcard characters, and dataout options are not permitted. The file attributes are:

- LCN file type—contiguous or linked
- LCN file protection
- Record size
- Block size
- LCN file configuration
- LCN file revision
- Directory timestamp (Mo/Dd/Yr Mm:Ss)
- Logical number of blocks
- Logical number of records
- File Descriptor
- Starting Sector
- Ending Sector

16.4.3.1 Example Pascal Call for File Attributes

```
Return_status  :=  CM50_ATTR_LIST
                     (lcn_file,
                      acidp_name,
                      attributes,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.3.2 Parameter Definitions for File Attributes

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (packed array of 28 characters declare as type LCN_PATH_NAME) identifying the file whose attributes are to be returned. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

attributes—Buffer (declare as FILE_ATTRIBUTE_BLOCK, and described in CM50$LIB:CM50_FTF_INCLUDE.pas) that will receive requested data.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.4 List LCN File Names

This transaction will retrieve up to 1180 file names and extensions from an LCN NET volume. If the number of files exceeds the buffer capacity of 1180 then multiple requests by directory, file type extension, or filename syntax must be used. Wildcards are permitted.

16.4.4.1 Example Pascal Call for List Files

```
Return_status  :=  CM50_FILE_LIST
                     (lcn_file,
                      acidp_name,
                      file_list,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.4.2 Parameter Definitions for List Files

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (packed array of 28 characters declare as type LCN_PATH_NAME) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:
NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

file_list—Buffer (declare as FILE_LIST_ARRAY, and described in CM50$LIB:CM50_FTF_INCLUDE.pas) that will receive the list of file names and attributes.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.5 List LCN Volumes/Directories

This transaction will fetch from the History Module volume and directory names and sector usage figures. Wildcards and options are not permitted for this transaction type.

16.4.5.1 Example Pascal Call for List Volumes

```
Return_status  :=  CM50_HM_LISTF
                     (lcn_device,
                      acidp_name,
                      vol_record,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.5.2 Parameter Definitions for List Volumes

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_device—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the device to be cataloged. Use the form PN:**nn** where **nn** is the lcn node number.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

vol_record—Buffer (declared as VOLUME_STRUCTURES, and described in CM50$LIB:CM50_FTF_INCLUDE.pas) that will receive the Volume and directory information. This information includes:

- Number of Volumes
- Number of Sectors / Device
- Sectors in Use / Device
- Volume Name(s)
- Directory Name(s) on each volume

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.6 Cataloging LCN Files to Dataout

This file transfer transaction will list the LCN FILE ATTRIBUTES of one or more files into the current dataout file. The dataout file must have been previously established. The absence of a dataout specification will result in an error return.

Further processing requires that the dataout or catalog file be transferred to the VAX using the CM50_LCN_READ programmatic function.

16.4.6.1 Example Pascal Call for File Catalog

```
Return_status :=  CM50_FILE_CATALOG
                     (lcn_file,
                      cat_file,
                      acidp_name,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.6.2 Parameter Definitions for File Catalog

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:

NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

Optional suffixes will increase the amount of information returned:

-FD will cause file descriptors to be listed
-REC will cause record and block data to be listed

cat_file—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the file to receive the catalog. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.7 Cataloging LCN Volumes to Dataout

This file transfer transaction will list the LCN Volumes and Directories for all History modules on the NET or for a specific History Module. The dataout file must have been previously established. The absence of a dataout specification will result in an error return.

16.4.7.1 Example Pascal Call for Volume Catalog

```
Return_status  :=  CM50_VOLUME_CATALOG
                     (lcn_device,
                      cat_file,
                      acidp_name,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.7.2 Parameter Definitions for Volume Catalog

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_device—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the device to be cataloged. Use the form NET or PN:**nn** where **nn** is the lcn node number.

cat_file—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the file to receive the catalog. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.8 LCN File Copy

This file transfer transaction will copy a single file or all files from one NET volume to another Net volume. The -D option is supported for journalizing all copies to a dataout file. The dataout file must have been previously established. Wildcards are permitted; however, the destination suffix must always be the same as the source suffix. Note that using the -D option without having previously defined a dataout path will result in an error and the copy function will not have been completed.

16.4.8.1 Example Pascal Call for LCN File Copy

```
Return_status  :=  CM50_LCN_COPY
                     (lcn_file,
                      out_file,
                      acidp_name,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.8.2 Parameter Definitions for LCN File Copy

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the file to be copied. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:
NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

out_file—Recipient filename (packed array of 28 characters, declare as type LCN_PATH_NAME) specifying the pathname of the new file. The actions will be journalized if a DATAOUT file has been enabled and the" -D" option suffix is appended to the filename.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.9 LCN File Move

This file transfer transaction will move a single file or all files from one directory to another directory within the same NET volume. Wildcards are permitted and the -D option is supported for journalizing all moves to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the move function will not have been completed.

16.4.9.1 Example Pascal Call for LCN File Move

```
Return_status  :=  CM50_LCN_MOVE
                     (lcn_file,
                      out_directory,
                      acidp_name,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.9.2 Parameter Definitions for LCN File Move

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the file to be moved. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:
NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

out_directory—Directory name (packed array of 28 characters, declare as type LCN_PATH_NAME) specifying the directory to receive the moved file. This directory must be on the same HM volume as the original file. (The file name and extensions will remain unchanged.) The actions will be journalized if a DATAOUT file has been enabled and the" -D" option suffix is appended to the filename.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.10 LCN File Rename

This file transfer transaction will rename a single file or all files on the History Module. Wildcards are permitted and the -D option is supported for journalizing all renames to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the rename function will not have been completed.

16.4.10.1 Example Pascal Call for LCN File Rename

```
Return_status  :=  CM50_LCN_RENAME
                     (lcn_file,
                      out_file,
                      acidp_name,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.10.2 Parameter Definitions for LCN File Rename

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the file to be renamed. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:
NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

out_file—Recipient filename (packed array of 28 characters, declare as type LCN_PATH_NAME) specifying the new file name. (The directory and extensions will remain unchanged.) The actions will be journalized if a DATAOUT file has been enabled and the " -D" option suffix is appended to the filename.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.11 LCN File Delete

This file transfer transaction will delete a single file or all files from the specified volume on the History Module. Wildcards are permitted and the -D option is supported for journalizing all deleted files to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the delete file function will not have been completed. Once deleted the file cannot be recovered.

16.4.11.1 Example Pascal Call for LCN File Delete

```
Return_status  :=  CM50_LCN_DELETE
                     (lcn_file,
                      acidp_name,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.11.2 Parameter Definitions for LCN File Delete

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the file to be copied. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:

NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

The actions will be journalized if a DATAOUT file has been enabled and the "-D" option suffix is appended to the pathname.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.12 LCN Directory Maintenance

These file transfer transactions will create or delete a directory under a volume on the History Module. No wildcard, characters or options are permitted.

16.4.12.1 Example Pascal Call for Directory Maintenance

```
Return_status := CM50_LCN_DIRECTORY
                        (lcn_directory,
                         action,
                         acidp_name,
                         cg_port_number,
                         lcn_sts,
                         flags);
```

16.4.12.2 Parameter Definitions for Directory Maintenance

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_directory—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the LCN directory to be created or deleted. Use the form NET>VDIR> DIR. (Note the space delimiter before the directory name.)

action—A shortword (declare as DIR_FUNC_TYPE) that specifies whether the named directory is to be created or deleted. The enumerated values are:

```
create_directory = 0
delete_directory = 1
```

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.13 Dataout Status

The dataout function allows the user, when requesting the execution of specific file transfer transactions, to place relative data in the dataout or catalog file. This dataout file is a shared file by all concurrent users of file transfer. For example, User "Jones" requests a CM50_FILE_CATALOG transaction, the results of which are placed into the current dataout file. User "Smith" then requests a CM50_VOLUME_CATALOG transaction. These results also are placed into the same or current dataout file. The dataout file may be transferred to the VAX host using a CM50_LCN_READ request. The CM50_DATA_OUT transaction is provided to enable, disable, or query the file transfer dataout status.

16.4.13.1 Example Pascal Call for DATAOUT status

```
Return_status  :=  CM50_DATA_OUT
                     (cat_file,
                      acidp_name,
                      do_action,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.13.2 Parameter Definitions for DATAOUT status

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

cat_file—LCN pathname (packed array of 28 characters, declare as type LCN_PATH_NAME) identifying the file to be used as the dataout journal. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

do_action—A shortword (declare as DO_FUNC_TYPE) that specifies the action to be taken. The values are:

`Disable` = 0: Disable dataout journaling
`Enable` = 1: = Enable dataout journaling using the specified path
`Request_status` =2: Return the current dataout path

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

16.4.14 ABORT FILE TRANSFER TRANSACTION

This transaction CM50_ABORT_TRANSFER will terminate the current transaction in progress. The initiator of the transaction will receive a CM50_FTF_ABORT error return status. The initiator of the CM50_ABORT_TRANSFER request will receive a normal return status. No error is generated if there is not a current process to abort.

16.4.14.1 Example Pascal Call for Abort Transfer

```
Return_status  :=  CM50_ABORT_TRANSFER
                     (acidp_name,
                      cg_port_number,
                      lcn_sts,
                      flags);
```

16.4.14.2 Parameter Definitions for Abort Transfer

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

acidp_name—A 16-character string (declare as ACIDPoint_Type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a shortword (declare as CG_NUM_TYPE) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a shortword (declare as INTV) which will receive the detailed error code from the LCN if the overall return_status is CM50_FTF_FILMGR (215004012) or CM50_FTF_UTILITY (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (declare as CM50_FLAG_TYPE) that sets options as described in section 13.1.3. The CM50$M_HANDLER option is the only flag applicable to File Transfer functions.

"C" LANGUAGE CONSIDERATIONS
Section 17

This section discusses each of the program interfaces that provide necessary services that enable "C" programs to communicate with other nodes on the TDC 3000 Local Control Network.

17.1 CM50S INCLUDE FILES

Each user interface routine has language-specific interfaces that are supported by include files that contain data declarations that match the argument names and symbolic constants used in the example calls in this section. **Any program that uses any of these interface routines should be compiled with the matching language-specific include files.**

17.1.1 Include Files for Data Transfer Functions

These include files will generally be used by Advanced Control Programs and Data Acquisition programs.

`CM50$LIB:CM50_INCLUDE.H`	Contains the declarations used by the LCN data interfaces (Sections 18 & 19) and the Vintage Routines (Appendix G).
`CM50$LIB:CM50_ERROR_INCLUDE.H`	Contains the symbolic names for all of the CM50S error codes (see Appendix A.2).
`CM50$LIB:CM50_FTF_INCLUDE.H`	Definitions for all LCN file transfer operations (section 20.4)

17.1.2 DDT and ACP Management

These include files are needed by applications that use the CM50S administration calls described in Section 20. DDT and ACP management functions use some shared data structures, that are defined in the file CM50_FLAGS_INCLUDE H. Therefore, that file should be included with any program that calls either DDT or ACP functions and must precede the include file defining those specific functions.

`CM50$LIB:CM50_FLAGS_INCLUDE.H`	Definitions for the shared data structures in the ACP & DDT Management Interface calls. Must be included prior to either CM50_ACP_INCLUDE.H or CM50_DDT_INCLUDE.H.
`CM50$LIB:CM50_ACP_INCLUDE.H`	Definitions for all ACP Management operations (section 20.1).
`CM50$LIB:CM50_DDT_INCLUDE.H`	Definitions for all the DDT Management operations (section 20.2).

CM50$LIB:CM50_CGDATA_INCLUDE.H	Definitions for all the CG Database retrievals (section 20.3).

17.1.3 Programmatic Interface Flag Parameters

An Integer parameter called **flags** is included in every ACP and DDT management function to control some of the handling options. Some of the flags apply to only the DDT calls, some to only the ACP calls, and some can be used by both. All user-visible flags (as defined in CM50_FLAGS_INCLUDE.H) are described below.

- cm50$m_handler—(Bit 0) Indicates that the user has provided a custom exception handler. The default is OFF.
- cm50$m_msgon—(Bit 1) Prints all diagnostic messages to SYS$OUTPUT. The default is OFF.
- cm50$m_cgres—(Bit 5) Installs the DDT as CG resident. The default is OFF.
- cm50$m_rebuild_ddt—(Bit 6) Rebuilds an existing DDT. The default is OFF.
- cm50$m_no_sourc_debug—(Bit 7) Produces no error file during DDT build. The default is OFF.
- cm50$m_dmp_ddt_errors—(Bit 8) Sends the error file produced by a DDT build to SYS$OUTPUT after building the DDT. If set, then the cm50$m_no_sourc_debug flag must be OFF.
- cm50$m_acidp_activate—(Bit 9) Reserved for internal CM50S use.
- cm50$m_write_vt—(Bit 10) Creates the .VT file with write privilege.

All of the flags described above, represent bit masks that can be added together to enable any combination of the flags. These flag values also can be used to see if a particular flag is set. An example is shown below.

```
 flags = cm50$m_handler + cm50$m_msgon;
Return_Status = ddt_summary(DDT_Name,
                            &Summary, &Flags)
```

17.2 CALLING CONVENTIONS

CM50S interface routines follow the VMS language-independent calling conventions. With the exception of some housekeeping procedures that have no error handling (such as ACPTRP and PRGTRM), they are written as functions.

We recommend that each function call be followed by a logical test of the return_status value. If return_status is odd valued (test as "`if (return_status &1)`"), the call was successful (although individual data items may require checking). Otherwise (even valued status codes), appropriate error handling should be invoked. Note that if the application does not check return_status, the interface routine can be invoked as a called procedure in the same manner as VMS system services.

Shortword arguments should be declared as **short int** and must be passed as variables because "C" assumes that any integer constant is a 32-bit integer.

Boolean (True/False) arguments are normally declared as short int (or type: cm50$bool2), with a value of 1 for True and 0 for False. ("C" evaluates all nonzero values as true, but 1 is the only acceptable true value to the LCN.)

A distinction is made between strings and character arrays. Arguments that are described as character arrays have a fixed size and are filled with trailing spaces. (The LCN does not recognize the null character as a field terminator.) Arguments that are described as strings may be passed as either null-terminated strings or as space-padded character arrays. **Using a null-terminated string in an argument declared as a character array will cause the function to return with an error.**

Character-string and array arguments are passed using the "C" default conventions. All other arguments must be explicitly passed by reference (using the "&" prefix). Strings and arrays may also by passed using the "&" qualifier. This may be desirable in applications that use arguments with dimensions other than those declared in the prototype declarations in the include files.

17.3 COMPATIBILITY OF APPLICATION PROGRAM WITH ITS DDTS

Because each application program and its Data Definition Tables (and Multi_Point List structures) are separately built, the system cannot enforce compatibility between a program and any DDT(s) that it uses. That responsibility is up to you.

In particular, it is vital that the dimensions set for data-receiving arrays be large enough to accommodate the maximum data amounts permitted by the named DDT.

Specific points to remember for DDT Get Data and DDT Store Data are

- Dimensions set for each value-type's program array must be equal-to or greater-than the value-type's point count in the referenced DDT. The values can be stored one-for-one or they can be scattered as defined in the DDT. If the program arrays are too small, data or program code may be corrupted (DDT Get Data) or inappropriate data may be exported (DDT Store Data).

- The dimension values for status table arrays must be equal-to or greater-than the total number of points of all types in the referenced DDT because this array is to receive a status code for each returned value, positioned according to its location in the DDT.

17.4 DATA REPRESENTATIONS

Differences between data representations in the VAX and the CG normally are resolved by the CG-VAX Communications Handlers, thus are invisible to the user (Exception: raw data transfers, see heading 18.3). The LCN data formats are:

Real—32 bit floating point matches normal float format except that bad values (NaN) from the LCN have the bit pattern for -0. This value will cause a VMS trap if used in an arithmetic statement, so real values returned from the LCN should always be tested (using either the CM50_VALIDN function or the associated value_status_table entry for the value).

Integer—short int.

ASCII—fixed-length array of 24 characters, padded with trailing spaces if necessary. (Not terminated by a null character.)

String—fixed-length array of 40 characters, padded with trailing spaces if necessary. (Not terminated by a null character.)

Enumeration—There are two ways to represent LCN enumerations: Enumerated ASCII as fixed-length arrays of 8 characters, padded with trailing spaces if necessary, (Not terminated by a null character); or Ordinal (short int) values. The choice of representation is made when the data transfer is requested, except that **self-defined enumerations should be transferred only as Ordinals**. For information on standard enumerations, see the *Application Module Parameter Reference Dictionary, Hiway Gateway Parameter Reference Dictionary*, and *Computer Gateway Parameter Reference Dictionary*. For information on Custom Data Segments, see the *System Control Functions* manual. For information on self-defined enumerations, see Section 2 of the *Hiway Gateway Control Functions* manual.

Time—LCN Internal Time is defined as a record structure (**cm50_time_vals**) consisting of an integer count of Seconds (since the start of 1979) followed by a short int count of Ticks (tenths of milliseconds). Some of the calls return LCN External Time as an array of 18 ASCII characters formatted as MM/DD/YYΔHH:MM:SSΔ, where Δ represents a space. See heading 19.3.3 for time format conversions.)

Entity ID—Internally stored as a 64-bit value (array of 4 short integers defined as type **cm50_ptid_vals**) identifying a specific point (Ptid or Internal_id). Also can be retrieved as an External_id, a fixed-length array of 18 characters, padded with trailing spaces if necessary. (Not terminated by a null character.) Note that the External_id array consists of the up-to-16-character point name followed by the 2-character pinid for Network Gateway references.

17.5 COMMONLY MADE ERRORS

- Character array arguments must be the declared length. If string constants are used for arguments, they must be padded with spaces. (Character string arguments may be of any appropriate length provided that they are terminated with a null character.)

- Failure to use the CM50_SET_ACP function (or ACPTRP call) as the first executable program statement of an ACP and/or failure to use the PRGTRM call as the last executable statement of an ACP.

- Attempting to run an application program with unresolved compile or link errors or the use of a DDT that is incomplete or complete with errors.

- Failure to specify array sizes and data types that match DDT definitions.

- Failure to specify all parameters required by the interface routines. Also, failure to specify the correct data type for parameters. Make sure the & prefix is used (to pass arguments by address instead of by value) everywhere it is shown in the examples.

- Attempting to activate an ACP through an ACIDP while the ACP is linked to the VMS DEBUG utility. Use of the DEBUG utility is supported only for execution of ACPs while run interactively from a terminal.

- Terminating an ACP by use of the STOP/IDENTIFIER function of VMS DCL. ACPs should only be aborted through the CM50 Deactivate ACP procedure.

17.6 ERROR DETECTION BY INTERFACE FUNCTIONS

There are three categories of error that can be detected during the execution of a program when using the interface functions. These are indicated through one of these methods:

- Request completion status code

- Individual parameter status codes

- Program abort

The RETURN_STATUS value returned by the Function shows whether or not the request was successfully processed and, if not, what error type was involved. Some typical errors flagged by the return status are

- LCN access problems or data link failure
- ACP installation or mode problems
- Data problems in the call or with a referenced DDT
- Call rules violations

The RETURN_STATUS code follows the standard VAX/VMS condition status code format. In general, even number codes indicate fatal system problems or program bugs, while odd number codes indicate success (code 000000001) or partial success (e.g., code 215000051). See Appendix A.2 for additional information and a listing of all RETURN_STATUS values and their meanings.

Most of the interface calls also return LCN point.parameter values that are to be processed by the calling application program. Accompanying each value (or array) is a status code that must be checked for indications of problems that would invalidate the requested data. See the call arguments STATUS_TABLE or VALUE_STATUS in the individual interface descriptions. There are over 200 different data access-status codes that can be returned. See Appendix A.1 for a listing of these codes.

Some errors in use of the interface routines result in the application program being aborted. An error message is logged at the VAX operator console and is shown on the Universal Station Detail Display for a connected ACIDP. These errors can be of the following types:

- File access errors
- Communication Interface errors
- Format conversion errors
- Various program logic errors

17.7 SUMMARY OF USER-PROGRAM INTERFACES

Heading	Interface Descriptions	Function Names
	Multipoint (DDT) Data Transfers	
18.1.1	DDT Get Data	cm50_ddt_get cm50_ddt_getnt
18.1.2	DDT Store Data	cm50_ddt_store cm50_ddt_storent
18.1.3	Generic DDT Get Data	cm50_ddt_getgen
18.1.4	Generic DDT Store Data	cm50_ddt_storegen
18.1.5	Multi-Point List Get Data	cm50_mpl_get
18.1.6	Multi-Point List Store Data	cm50_mpl_store
18.1.7	Generate Multi-Point List	cm50_mpl_genlist cm50_mpl_gentags cm50_mpl_genfile
18.1.8	Read Multi-Point List	cm50_mpl_read
18.1.9	Write Multi-Point List	cm50_mpl_write
18.1.10	Create Include File for Multi-Point List	cm50_mpl_genincl
	Point List Data Transfers	
18.2.1	Point List Get Values	cm50_get_pt_list
18.2.2	Point List Get by Value Type	
	Real Values	cm50_get_realnbr
	Integer Values	cm50_get_intnbr
	ASCII Values	cm50_get_asc24
	Enumeration Values	cm50_get_enum
	Ordinal Values	cm50_get_ord
	Internal Ids	cm50_get_ptid
	External Ids	cm50_get_exid
	Time Values	cm50_get_time
	String Values	cm50_get_stri
18.2.3	Point List Store Values	cm50_store_pt_list
18.2.4	Point List Store by Value Type	
	Real Values	cm50_store_realnbr
	Integer Values	cm50_store_intnbr
	ASCII Values	cm50_store_asc24
	Enumeration Values	cm50_store_enum
	Ordinal Values	cm50_store_ord
	Internal Ids	cm50_store_ptid
	Time Values	cm50_store_time
	String Values	cm50_store_stri
	Single Point Data Transfers	
18.3.1	Single Point Get Data(External ID)	cm50_get_id cm50_get_tag
18.3.2	Single Point Store Data(External ID)	cm50_store_id cm50_store_tag
18.3.3	Single Point Get Data (Internal ID)	cm50_getpt_id
18.3.4	Single Point Store Data (Internal ID)	cm50_storept_id
18.3.5	Get lcn Clock Value	cm50_timnow_lcn cm50_timnow_asc
	Raw Data Transfers	
18.4.1	Raw Data Get	cm50_spgraw
18.4.2	Raw Data Store	cm50_spsraw
18.4.3	Convert Raw Data	cm50_spcraw

Heading	Interface Descriptions	Function Names
	History Data Transfers	
18.5.2	Get History Snapshots (Relative Time)	cm50_ddthis_snap cm50_ddthis_fast cm50_mplhis_snap cm50_pthis_snap
18.5.3	Get History Snapshots (Absolute Time)	cm50_ddthis_snapt cm50_ddthis_fastt cm50_mplhis_snapt cm50_pthis_snapt
18.5.4	Get History Averages (Relative Time)	cm50_ddthis_aver cm50_mplhis_aver cm50_pthis_aver
18.5.5	Get History Averages (Absolute Time)	cm50_ddthis_avert cm50_mplhis_avert cm50_pthis_avert
18.5.6	Get Monthly Averages (Relative Time)	cm50_ddthis_mnth cm50_mplhis_mnth cm50_pthis_mnth
18.5.7	Get Monthly Averages (Absolute Time)	cm50_ddthis_mntht cm50_mplhis_mntht cm50_pthis_mntht
18.5.8	Find History Collection Rate	cm50_ddthis_rate cm50_mplhis_rate cm50_pthis_rate
	Text Message Transfers	
18.6.1	Get Message	cm50_getmsg
18.6.2	Send Message	cm50_storemsg
	ACP Execution Support	
19.1.1	ACP Initializaton	cm50_set_acp
19.1.2	Get ACP Status	getsts*
19.1.3	ACP Delay	cm50_acpdelay
19.1.4	ACP Hibernate	cm50_hiber
19.1.5	ACP Termination	prgtrm*
	Entity Name Conversions	
19.2.1	Convert External to Internal ID	cm50_conv_pt cm50_conv_tag
19.2.2	Convert List of External Ids	cm50_conv_pt_list cm50_conv_tag_list
	Value Conversions	
19.3.1	Valid Number Check	cm50_validn
19.3.2	Set Bad Value	cm50_setbad
19.3.3	Convert Time Values	cm50_timlcn_ary cm50_timlcn_asc cm50_timlcn_euro cm50_timlcn_vaxa cm50_timlcn_vaxb cm50_timary_lcn cm50_timary_asc cm50_timary_euro cm50_timary_vaxa cm50_timary_vaxb cm50_timasc_lcn

* GETSTS and PRGTRM do not have a RETURN_STATUS, so they cannot be used as functions, but must be invoked as procedures.

Heading	Interface Descriptions	Function Names
19.3.3	Convert Time Values—continued	cm50_timasc_ary cm50_timasc_euro cm50_timasc_vaxa cm50_timasc_vaxb cm50_timeuro_lcn cm50_timeuro_ary cm50_timeuro_asc cm50_timeuro_vaxa cm50_timeuro_vaxb cm50_timvaxa_lcn cm50_timvaxa_ary cm50_timvaxa_asc cm50_timvaxa_euro cm50_timvaxa_vaxb cm50_timvaxb_lcn cm50_timvaxb_ary cm50_timvaxb_asc cm50_timvaxb_euro cm50_timvaxb_vaxa
	ACP Management	
20.1.1	Install an ACP	cm50_acp_install
20.1.2	Uninstall an ACP	cm50_acp_uninst
20.1.3	Activate (run) an ACP	cm50_acp_act
20.1.4	Deactivate (abort) an ACP	cm50_acp_deactivate
20.1.5	Connect an ACP to an ACIDP	cm50_acp_connect
20.1.6	Disconnect ACP from its ACIDP	cm50_acp_discon
20.1.7	Change ACP installation mode	cm50_acp_chg_mode
20.1.8	Get ACP summary	cm50_acp_sum
20.1.9	Get list of ACPs	cm50_acp_listall
	DDT Management	
20.2.1	Build/Rebuild a DDT	cm50_ddt_build
20.2.2	Delete a DDT	cm50_ddt_delete
20.2.3	Get DDT summary information	cm50_ddt_sum
20.2.4	Get list of DDT summaries	cm50_ddt_list
20.2.5	Get DDT detailed information	cm50_ddt_detail
20.2.6	Connect a DDT to an ACIDP	cm50_ddt_connect
20.2.7	Disconnect a DDT from its ACIDP	cm50_ddt_disconnect
20.2.8	Modify DDT prefetch triggers	cm50_ddt_triggers
20.2.9	Install a DDT as CG resident	cm50_ddt_install
20.2.10	Remove a DDT from CG residency	cm50_ddt_uninst
	CG Database Routines	
20.3.1	Get list of resident DDTs	cm50_cg_rddt
20.3.2	Get list of CRDPs	cm50_cg_crdp
20.3.3	Get detailed ACIDP information	cm50_cg_adetail
20.3.4	Get list of ACIDPs	cm50_cg_acidp
20.3.5	Get LCN Configuration	cm50_cg_config
	File Transfer Routines	
20.4.1	Read File from LCN	cm50_lcn_read
20.4.2	Write File to LCN	cm50_lcn_write
20.4.3	List LCN File Attributes	cm50_attr_list
20.4.4	List LCN Files & Extensions	cm50_file_list
20.4.5	List LCN Volumes/Directories	cm50_HM_list

20.4.6	List LCN Files to Dataout	cm50_file_catalog
20.4.7	List LCN Volumes to Dataout	cm50_volume_catalog
20.4.8	LCN File Copy	cm50_lcn_copy
20.4.9	LCN File Move	cm50_lcn_move
20.4.10	LCN File Rename	cm50_lcn_rename
20.4.11	LCN File Delete	cm50_lcn_delete
20.4.12	LCN Directory Maintenance	cm50_lcn_directory
20.4.13	LCN Dataout Status	cm50_data_out
20.4.14	Abort LCN File Transfer	cm50_abort_transfer

LCN DATA TRANSFERS (“C”)
Section 18

This section discusses each of the program calls that "C"programs use to transfer data between the host computer and the TDC 3000 Local Control Network.

18.1 MULTIPOINT (DDT) DATA TRANSFERS

The interface routines in this group require the use of separately prepared Data Definition Tables (DDT) that specify which points are to be accessed and what pre/post processing is to be done on data values. See Section 6 for DDT preparation and installation details.

Each DDT may reference a maximum of four different data types. The standard DDT functions assume the data types are grouped into a "normal" order. It is possible to build DDTs with unusual combinations of data types that do not follow these assumptions. These special-case DDTs are tagged as GenIn (Generic Input) or GenOut (Generic Output) and may only be used with the Generic DDT Transfers described in subsections 18.1.3 and 18.1.4. Standard Input and Output DDTs may be used with either the Generic DDT transfers or the traditional DDT data interface routines.

Single elements of parameter arrays (but not whole arrays) can be specified in the DDT.

18.1.1 DDT Get Data Interface

This routine fetches data from the DDT's associated CG or elsewhere on its LCN. The specification of which data is to be fetched and where it is to be stored in the calling program's data arrays is contained in the Data Definition Table referenced by the call.

18.1.1.1 Example "C" Call for DDT Get Data

```
return_status =  cm50_ddt_get  or cm50_ddt_getnt
                 (ddt_name,
                  real_values_array,
                  intg_values_array,
                   or  &ptid_values_array,
                   or  &time_values_array,
                  asci_values_array,
                   or  &string_values_array,
                   or  &exid_values_array,
                  enum_array,
                   or  &ord_array,
                  status_table);
```

Use the interface name `CM50_DDT_GET` if you want data transformation operations performed by the Table Processor, and `CM50_DDT_GETNT` if you do not want data transformation operations performed (to decrease processing time). The DDT Get Data

call must specify four data types in the order shown (three of these can be dummy arguments that receive no data). Note that there are restrictions on data-type combinations.

18.1.1.2 Parameter Definitions for DDT Get Data

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (cm50_lcn_part), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a string of 9 characters (prototyped as type: cm50_ddt_name_type) that contains the name of the input Data Definition Table to be used.

real_values_array—The name of an array of up to 300 float numbers (prototyped as type: cm50$real300) where the fetched Real values are to be stored. Bad values are returned as NaN (-0).

intg_values_array—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) where the fetched Integer values are to be stored.

ptid_values_array—The name of an array of up to 300 internal entity ids (prototyped as type: cm50_ptid_vals).

time_values_array—The name of an array of up to 300 LCN internal time values (prototyped as type: cm50_time_array_type).

asci_values_array—The name of an array of 24-character arrays (prototyped as type: cm50_asci_array_type) where the fetched ASCII values are to be stored. Bad values are returned as strings of question marks.

string_values_array—The name of an array of up to 300 40-character arrays (prototyped as type: cm50_stri_array_type) where the fetched LCN string values are to be stored.

exid_values_array—The name of an array of up to 300 18-character arrays (prototyped as type: cm50_exid_array_type) where the fetched external entity names are to be stored.

enum_array—The name of an array of up to 300 8-character arrays (prototyped as type: cm50_aenm_array_type) where the fetched Enumeration values are to be stored. Bad values are returned as strings of question marks.

ord_array—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) where the fetched ordinal values of enumerations are to be stored.

status_table—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A.1 for a listing of Data Access error/status codes.

18.1.2 DDT Store Data Interface

This routine sends data to points in the DDT's associated CG or elsewhere on its LCN. The specification of what points are to receive data and the location of data within the calling program's data arrays is contained in the Data Definition Table referenced by the call. Errors encountered during execution of the routine as well as individual point-data errors are returned to the calling program.

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

18.1.2.1 Example "C" Call for DDT Store Data

```
return_status =  cm50_ddt_store  or  cm50_ddt_storent
                 (ddt_name,
                  real_values_array,
                  intg_values_array,
                   or  &ptid_values_array,
                   or  &time_values_array,
                  asci_values_array,
                   or  &string_values_array,
                  enum_array,
                   or  &ord_array,
                  store_array,
                  status_table);
```

Use the Interface Name CM50_DDT_STORE if you want data transformation operations performed by the Table Processor and CM50_DDT_STORENT if you do not want transformation operations performed (to decrease processing time).

The DDT Store Data call must specify four data types in the order shown (three of these can be dummy arguments that export no data). Note that there are restrictions on the combinations of data type.

18.1.2.2 Parameter Definitions for DDT Store Data

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (cm50_lcn_part), which indicates that the `status_table` status code for each requested store value must be checked.

ddt_name—The name of a string of 9 characters (prototyped as type: cm50_ddt_name_type) that contains the name of the output Data Definition Table to be used in the "Store Data" operation.

real_values_array—The name of an array of up to 300 float numbers (prototyped as type: cm50$real300) containing the Real values to be stored.

intg_values_array—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) containing the Integer values to be stored.

ptid_values_array—The name of an array of up to 300 internal entity ids (prototyped as type: cm50_ptid_vals).

time_values_array—The name of an array of up to 300 LCN internal time values (prototyped as type: cm50_time_array_type).

asci_values_array—The name of an array of 24-character arrays (prototyped as type: cm50_asci_array_type) containing the ASCII values to be stored.

string_values_array—The name of an array of up to 300 40-character arrays (prototyped as type: cm50_stri_array_type) containing the LCN string values to be stored.

enum_array—The name of an array of up to 300 8-character arrays (prototyped as type: cm50_aenm_array_type) containing the Enumeration values to be stored.

ord_array—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) containing the ordinal values of enumerations to be stored.

store_array—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) that contains a control code entry for each value to be stored. These codes control what—if any—value is to be stored. The store code values are

0 – Store the value from the Values Array
1 – Store the bad value representation instead
2 – Do not store any value.

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

status_table—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A for a listing of Data Access error/status codes.

18.1.3 Generic DDT Get Data Interface

This routine fetches data for any Input or Generic Input DDT. The specification of which data is to be fetched and where it is to be stored in the calling program's data arrays is contained in the Data Definition Table referenced by the call.

18.1.3.1 Example "C" Call for Generic DDT Get

```
return_status =  cm50_ddt_getgen
                 (ddt_name,
                  &values_array1,
                  &values_array2,
                  &values_array3,
                  &values_array4,
                  status_table,
                  &tbl_proc);
```

18.1.3.2 Parameter Definitions for Generic DDT Get

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (cm50_lcn_part), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a string of 9 characters (prototyped as type: cm50_ddt_name_type) that contains the name of the output Data Definition Table to be used in the "Store Data" operation.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array where the fetched values are to be stored. The data type for each array must match the corresponding data type in the DDT definition. In "C", each array may be declared to match the specific type and number of elements returned by the DDT; if the DDT contains fewer than 4 data types, the unused arguments may be omitted (but the correct number of commas is required).

status_table—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A.1 for a listing of Data Access error/status codes.

tbl_proc—The name of a short integer that determines whether or not table processing is to be suppressed. If tbl_proc is set to 1, all table processing (saving values to disk and/or data transformations) will be suppressed. Use a value of 0 for normal processing.

18.1.4 Generic DDT Store Data Interface

This routine sends data to points defined in any Output or Generic Output DDT. The specification of what points are to receive data and the location of data within the calling program's data arrays is contained in the Data Definition Table referenced by the call. Errors encountered during execution of the routine as well as individual point-data errors are returned to the calling program.

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

18.1.4.1 Example "C" Call for Generic DDT Store

```
return_status =  cm50_ddt_storegen
                 (ddt_name,
                  &values_array1,
                  &values_array2,
                  &values_array3,
                  &values_array4,
                  store_array,
                  status_table,
                  &tbl_proc);
```

18.1.4.2 Parameter Definitions for Generic DDT Store

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (cm50_lcn_part), which indicates that the `status_table` status code for each requested store value must be checked.

ddt_name—The name of a string of 9 characters (prototyped as type: cm50_ddt_name_type) that contains the name of the output Data Definition Table to be used in the "Store Data" operation.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array that contains the data to be stored. The data type for each array must match the corresponding data type in the DDT definition. In "C", each array may be declared to match the specific type and number of elements returned by the DDT; if the DDT contains fewer than 4 data types, the unused arguments may be omitted (but the correct number of commas is required).

store_array—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) that contains a control code entry for each value to be stored. These codes control what—if any—value is to be stored. The store code values are

- 0 – Store the value from the Values Array
- 1 – Store the bad value representation instead
- 2 – Do not store any value.

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

status_table—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag (in the same order as the DDT source file). See Appendix A for a listing of Data Access error/status codes.

tbl_proc—The name of a short integer that determines whether or not table processing is to be suppressed. If tbl_proc is set to 1, all table processing (saving values to disk and/or data transformations) will be suppressed. Use a value of 0 for normal processing.

18.1.5 Multi-Point List Get Data Interface

This routine fetches data for the LCN tags specified in an internal data block. An internal Data Block is a memory-resident equivalent of a DDT. The specification of which data is to be fetched and where it is to be stored in the calling program's data arrays can be prepared using any of the generate MPL routines (see 18.1.7) or you can read in a DDT from its disk file (see 18.1.8).

18.1.5.1 Example "C" Call for Multi-Point List Get

```
return_status =  cm50_mpl_get
                 (&mpl_name,
                  acidp_name,
                  &values_array1,
                  &values_array2,
                  &values_array3,
                  &values_array4,
                  status_table,
                  &cg_port_num);
```

18.1.5.2 Parameter Definitions for Multi-Point List Get

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (cm50_acp_run)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (cm50_lcn_part)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

mpl_name—The name of a Multi-Point List structure defining the data to be retrieved. This should be declared as a record of type cm50_idb_rec.

acidp_name—A string of 16 characters (prototyped as type: cm50_long_acidp) containing the name of an ACIDP. If the ACIDP is spaces, then the data will be retrieved without any ACIDP controls. If an ACIDP is named, then the data access will be completed only if that ACIDP is in RUN state.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array where the fetched values are to be stored. The data type for each array must match the corresponding data type in the MPL definition. In "C", each array may be declared to match the specific type and number of elements returned by the MPL; if the MPL contains fewer than 4 data types, the unused arguments may be omitted (but the correct number of commas is required).

status_table—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag in the list. See Appendix A.1 for a listing of Data Access error/status codes.

cg_port_num—The name of a short integer (with a value of 1-4) identifying the CG to be accessed.

18.1.6 Multi-Point List Store Data Interface

This routine stores data for the LCN tags specified in an internal data block. An internal Data Block is a memory-resident equivalent of a DDT. The specification of which tags are to receive data and the location of the values within the calling program's data arrays can be prepared using any of the generate MPL routines (see 18.1.7) or you can read in a DDT from its disk file (see 18.1.8).

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

18.1.6.1 Example "C" Call for Multi-Point List Store

```
return_status =  cm50_mpl_store
                 (&mpl_name,
                  acidp_name,
                  &values_array1,
                  &values_array2,
                  &values_array3,
                  &values_array4,
                  store_array,
                  status_table,
                  &cg_port_num);
```

18.1.6.2 Parameter Definitions for Multi-Point List Store

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (cm50_acp_run)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (cm50_lcn_part)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

mpl_name—The name of an Multi-Point List structure defining the data to be stored. This should be declared as a record of type cm50_idb_rec.

acidp_name—A string of 16 characters containing the name of an ACIDP. If the ACIDP is spaces, then the ACIDP currently connected to the ACP will control the data transfer. If an ACIDP is named, then the data access will be completed only if that ACIDP is in RUN state.

values_array*n*—(where ***n*** is 1, 2, 3 or 4) The name of an array that contains the data to be stored. The data type for each array must match the corresponding data type in the MPL definition. Each array may be declared to match the specific type and number of elements returned by the MPL; if the MPL contains fewer than 4 data types, the unused arguments may be omitted (but the correct number of commas is required).

store_array—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) that contains a control code entry for each value to be stored. These codes control what—if any—value is to be stored. The store code values are

0 – Store the value from the Values Array
1 – Store the bad value representation instead
2 – Do not store any value
16386 - Store IEEE negative infinity instead of Real value
16387 - Store IEEE positive infinity instead of Real value

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

status_table—The name of an array of up to 300 short integers (prototyped as type: cm50$int300) for the storage of returned point-related error/status information. A value_status code is returned for each requested tag in the list. See Appendix A.1 for a listing of Data Access error/status codes.

cg_port_num—The name of a short integer (with a value of 1-4) identifying the CG to be accessed.

18.1.7 Generate Multi-Point List

These routines generate an Internal data block for transfer arrays of up to four data types between the LCN and host computer. Internal data blocks are subject to exactly the same restrictions as DDTs (see Table 6-1).

A Multi-Point List may be generated from either a set of ID Block Arrays (such as those produced using the Convert Lists calls—see section 19.2.2), or a text file of type declarations and tag names, or an array of text entries.

NOTE

The arrays of internal point.parameter addresses need to be rebuilt and the program(s) using them need to be recompiled whenever the LCN database is changed in a significant manner, such as by the rebuild or deletion of data points referenced in the address array.

18.1.7.1 Example "C" Calls to Generate Multi-Point Lists

To combine point lists, use:

```
return_status =     cm50_mpl_genlist
                    (list_size,
                     id_block_arr1,
                     id_block_arr2,
                     id_block_arr3,
                     id_block_arr4,
                     &mpl_name);
```

When the external ids are expressed as a Tag name list, use:

```
return_status =     cm50_mpl_gentags
                    (tagname_arr,
                     &number_of_values,
                     &mpl_name,
                     &cg_port_num
                     return_arr);
```

When the external ids are contained in a Text file, use:

```
return_status =     cm50_mpl_genfile
                    (tag_file,
                     &mpl_name,
                     &cg_port_num
                     return_arr);
```

18.1.7.2 Parameter Definitions for Generate Multi-Point Lists

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_LCN_PART), which indicates that the `return_arr` status code for each returned value must be checked.

tagname_arr—The name of an array of up to 304 strings of 40 characters (prototyped as type: cm50_tag_list_type) that contains the ASCII Tagname (formatted as Point.Param, optionally with the parameter index enclosed in parentheses) of the LCN entity for which the internal ID is to be obtained. All tags of the same data type must be grouped together and different data types must be separated by the reserved "tag" of: `**NEWΔTYPE=type`
where "Δ" is a required space and "type" (starting in position 11) is one of the following:

REAL	float number
INTE	16-bit integer
ASCI	24 character ASCII
ENUM	enumeration
ORDN	ordinal
PTID	internal entity id
EXID	external entity id
TIME	lcn time type
STRI	40 character string

If the first item in the array does not contain a "**NEW TYPE=" in positions 0 through 10, then the first set of tags is assumed to identify Real numbers.

number_of_values —The name of a short integer specifying the number of tags defined in the id_block_arr. The maximum number of values is 304.

tag_file—A string of 80 characters (prototyped as type: cm50_file_name_type) that names a text file whose content is a tagname_array, with each line containing either a valid tagname or a "**NEW TYPE=" tag as described above.

list_size —The name of an array of 4 short integers (prototyped as type: cm50_ptid_vals) specifying the number of tags defined in each id_block_arr. The maximum number of values is 300.

id_block_arr*n* —(where **n** is 1 to 4) The name of a point list array (declared as cm50_point_list_array_type) which may combine up to 4 different data different types, with a maximum of 300 16-byte variables. If multiple data types are included, then all entries of the same type must be grouped together. The size of the point list must match that specified in list_size[**n**]. If there are fewer than 4 data types, the unused arguments may be omitted (but the correct number of commas is required).

mpl_name—The name of an Multi-Point List structure where the generated definition is to be stored. This should be declared as a record of type cm50_idb_rec.

cg_port_num—The name of a short integer (with a value of 1-4) identifying the CG to be accessed.

return_arr—The name of an array of up to 304 integers (prototyped as type: cm50_return_arr_type) that receives the status of the conversion of each tag and data type declaration, including field type records. See Appendix A.2 for an explanation and a listing of all assigned return code values.

18.1.8 Read Multi-Point List

This routine reads an MPL from a disk file that has been created using either the DDT Build procedures or the Write Multi-Point List routine.

18.1.8.1 Example "C" Calls to Read Multi-Point Lists

```
return_status =     cm50_mpl_read
                    (idb_file,
                     &mpl_name);
```

18.1.8.2 Parameter Definitions for Read Multi-Point List

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

idb_file—A string of 80 characters (prototyped as type: cm50_file_name_type) that names the path to a file containing the Multi-Point List. To reference a DDT, use the pathname of `CM50$DDT:ddtname.II`. If no extension is specified, the default of .MPL will be used.

mpl_name—The name of a Multi-Point List structure in memory. This should be declared as a record of type cm50_idb_rec.

18.1.9 Write Multi-Point List

This routine creates a disk file containing an MPL produced through the Generate Multi-Point List interface (section 18.1.7).

18.1.9.1 Example "C" Calls to Write Multi-Point Lists

```
return_status =     cm50_mpl_write
                    (idb_file,
                     &mpl_name);
```

18.1.9.2 Parameter Definitions for Write Multi-Point List

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

idb_file—A string of 80 characters (prototyped as cm50_file_name_type) that names the path to a file to contain the Multi-Point List. If a file by that name already exists, a new version will be created. By default, an extension of .MPL will be used. The use of .II as an extension is prohibited because that extension is reserved for DDTs. It is the user's responsibility to purge obsolete versions.

mpl_name—The name of an Multi-Point List structure in memory. This should be declared as a record of type cm50_idb_rec.

18.1.10 Create Include File for Multi-Point List

This routine creates a disk file containing the text description of an MPL in a format suitable for use as an include file for a "C" source program. The MPL should be previously produced through the Generate Multi-Point List interface (see heading 18.1.7).

18.1.18.1 Example "C" Call to Generate a Multi-Point List Include File

```
return_status =     cm50_mpl_genincl
                    (&mpl_name,
                     text_file,
                     &language);
```

18.1.18.2 Parameter Definitions for Generate Multi-Point List Include File

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

mpl_name—The name of an Multi-Point List structure in memory. This should be declared as a record of type cm50_idb_rec.

text_file—A string of 80 characters (prototyped as cm50_file_name_type) that names the path of the include file to be written. If a file by that name already exists, a new version will be created. No default extension is provided. It is the users responsibility to purge obsolete versions.

language— A single character (declared as char) code identifying the format of the include file:

'P' = Pascal
'C' = C
'F' = FORTRAN

Any other value will default to FORTRAN.

18.2 POINT LIST TRANSFERS

These routines enable you to address multiple points with a single call without the necessity to build DDT tables. In the place of a DDT reference, you will have to provide a pointer to an array of "internal" point.parameter addresses. These internal addresses can be obtained by conversion calls at program runtime (see heading 19.2), or in advance by creating an include file through the Utility MAKEINC (see heading 7.2).

18.2.1 Point List Get Values Interface

This function returns data values to up-to-300 points on the LCN without using DDT tables. The specification of which data is to be fetched and where it is to be stored is contained in the call.

Use of Internal Point-parameter IDs is required. Individual elements of parameter arrays can be specified by repeating the point.parameter address using a changed parameter index. The data type of the values is determined from the Internal ID of the first point in the list.

18.2.1.1 Example "C" Call for Point List Get Values

```
return_status =  cm50_get_pt_list
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  &values_array,
                  status_table,
                  &number_of_values);
```

18.2.1.2 Parameter Definitions for Point List Get Real Values

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (cm50_acp_run)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (cm50_lcn_part)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

cg_port_num—The name of a short integer (prototyped as cm50$uword) identifying the CG (1-4) to be accessed.

priority—The name of a short integer (prototyped as cm50$uword) that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

acidp_name—The name of a 16-character string (prototyped as cm50_long_acidp) that contains the name of an ACIDP. If the ACIDP name value is blank (all spaces), then the data is retrieved without any ACIDP controls. If an ACIDP is named, then the data access is completed only if that ACIDP is in RUN state.

point_list_array—The name of an array of point addresses in internal format (declare as an array of up to 300 cm50_idblk records) See the function Convert External to Internal ID functions (heading 19.2) for additional information.

values_array—The name of an array of up to 300 values (whose type is compatible with the requested value type) containing the individual values to be stored.

status_table—The name of an array of up to 300 short integers (prototyped as cm50$int300) where the value status for individual point values are to be stored. See Appendix A.1 for a listing of Data Access error/status codes.

number_of_values—The name of a short integer (prototyped as cm50$int2) that specifies the actual number of values (300 or less) to be processed.

18.2.2 Point List Get By Value Type

These functions are identical to the CM50_GET_PT_LIST function, except that the value type is part of the function name and the generic "values_array" argument is replaced by an an array whose data type explicitly matches the specified data type.

This specific functions and their corresponding value arrays are described below. Refer to heading 18.2.1.2 for explanations of all of the other arguments.

18.2.2.1 "C" Call for Point List Get Real Values

```
return_status =  cm50_get_realnbr
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  real_values_array,
                  status_table,
                  &number_of_values);
```

real_values_array—The name of an array of up to 300 float numbers (prototyped as cm50$real300) where the individual point values are to be stored.

18.2.2.2 "C" Call for Point List Get Integer Values

```
return_status =  cm50_get_intnbr
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  intg_values_array,
                  status_table,
                  &number_of_values);
```

intg_values_array—The name of an array of up to 300 short integer (prototyped as cm50$int300) where the individual point values are to be stored.

18.2.2.3 "C" Call for Point List Get ASCII Values

```
return_status =  cm50_get_asc24
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  asci_values_array,
                  status_table,
                  &number_of_values);
```

asci_values_array—The name of an array of up to 300 fixed-length 24-character arrays (prototyped as cm50$asci300) where the individual point values are to be stored.

18.2.2.4 "C" Call for Point List Get Enumerated Values

```
return_status =  cm50_get_enum
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  aenm_values_array,
                  status_table,
                  &number_of_values);
```

aenm_values_array—The name of an array of up to 300 fixed-length 8-character arrays (prototyped as cm50$enum300) where the individual point values are to be stored.

18.2.2.5 "C" Call for Point List Get Ordinal Values

```
return_status =  cm50_get_ord
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  oenm_values_array,
                  status_table,
                  &number_of_values);
```

oenm_values_array—The name of an array of up to 300 short integers (prototyped as cm50$int300) where the ordinal values of the fetched enumerations are to be stored.

18.2.2.6 "C" Call for Point List Get Internal IDs

```
return_status =  cm50_get_ptid
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  ptid_values_array,
                  status_table,
                  &number_of_values);
```

ptid_values_array—The name of an array of up to 300 64-bit internal entity ids (prototyped as cm50_idblk) where the individual point values are to be stored.

18.2.2.7 "C" Call for Point List Get External IDs Values

```
return_status =  cm50_get_exid
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  exid_values_array,
                  status_table,
                  &number_of_values);
```

exid_values_array—The name of an array of up to 300 fixed-length 18-character arrays (prototyped as cm50$exid_vals) where the individual point values are to be stored.

18.2.2.8 "C" Call for Point List Get Time Values

```
return_status =  cm50_get_time
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  time_values_array,
                  status_table,
                  &number_of_values);
```

time_values_array—The name of an array of up to 300 cm50_time_vals records where the individual point values are to be stored.

18.2.2.9 "C" Call for Point List Get String Values

```
return_status =  cm50_get_stri
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  stri_values_array,
                  status_table,
                  &number_of_values);
```

stri_values_array—The name of an array of up to 300 fixed-length 40-character arrays (prototyped as cm50_stri_vals) where the individual point values are to be stored.

18.2.3 Point List Store Values Interface

This function exports data values to up-to-300 points on the LCN without using DDT tables. The specification of which data is to be fetched and where it is to be stored is contained in the call.

Use of Internal Point-parameter IDs is required. Individual elements of parameter arrays can be specified by repeating the point.parameter address using a changed parameter index. The data type of the values is determined from the Internal Id of the first point in the list. Note that entity ids can only be stored using their internal form.

18.2.3.1 Example "C" Call for Point List Store Values

```
return_status =  cm50_store_pt_list
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  &values_array,
                  store_code_table,
                  status_table,
                  &number_of_values);
```

18.2.3.2 Parameter Definitions for Array Store Values

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000042 (cm50_acp_run)—The data access could not be completed because the specified ACIDP is not in RUN state; Indirect Control programs should retry (indicates contention for ACIDP).

215000051 (cm50_lcn_part)—The returned data contains errors, thus the **status_table** status code for each returned value must be checked.

cg_port_num—The name of a short integer (prototyped as cm50$uword) identifying the CG (1-4) to be accessed.

priority—The name of a short integer (prototyped as cm50$uword) that contains the requested data-access priority:
1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

acidp_name—The name of a 16-character string (prototyped as cm50_long_acidp) that contains the name of an ACIDP. If the ACIDP name value is blank (all spaces), then the ACIDP currently connected to the ACP will control the data transfer. The data access is completed only if the named or implied ACIDP is in RUN state.

point_list_array—The name of an array of point addresses in internal format (declare as an array of up to 300 cm50_idblk records) See the function Convert External to Internal ID functions (heading 19.2) for additional information.

values_array—The name of an array of up to 300 values (whose type is compatible with the requested value type) containing the individual values to be stored.

store_code_table—The name of an array of up to 300 short integers (prototyped as cm50$int300) array where the calling program has stored a control code for each value to be stored. These codes control what—if any—value is to be stored. The store code values are:

0 = Store the value from the Real Values Array
1 = Store the bad value representation (NaN) instead
2 = Do not store any value
16386 = Store IEEE negative infinity instead of Real value
16387 = Store IEEE positive infinity instead of Real value

status_table—The name of an array of up to 300 short integers (prototyped as cm50$int300) where the value status for individual point values are to be stored. See Appendix A.1 for a listing of Data Access error/status codes.

number_of_values—The name of a short integer (prototyped as cm50$int2) that specifies the actual number of values (300 or less) to be processed.

18.2.4 Point List Store By Value Type

These functions are identical to the CM50_STORE_PT_LIST function, except that the value type is part of the function name and the generic "values_array" argument is replaced by an an array whose data type explicitly matches the specified data type.

This specific functions and their corresponding value arrays are described below. Refer to heading 18.2.3.2 for explanations of all of the other arguments.

18.2.4.1 "C" Call for Point List Store Real Values

```
return_status =  cm50_store_realnbr
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  real_values_array,
                  store_code_table,
                  status_table,
                  &number_of_values);
```

real_values_array—The name of an array of up to 300 float values (prototyped as cm50$real300) containing the individual values to be stored.

18.2.4.2 "C" Call for Point List Store Integer Values

```
return_status =  cm50_store_intnbr
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  intg_values_array,
                  store_code_table,
                  status_table,
                  &number_of_values);
```

intg_values_array—The name of an array of up to 300 short integers (prototyped as cm50$int300) containing the individual values to be stored.

18.2.4.3 "C" Call for Point List Store ASCII Values

```
return_status =  cm50_store_asc24
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  asci_values_array,
                  store_code_table,
                  status_table,
                  &number_of_values);
```

asci_values_array—The name of an array of up to 300 fixed-length 24-character arrays (prototyped as cm50$asci300) containing the individual point values to be stored.

18.2.4.4 "C" Call for Point List Store Enumerated Values

```
return_status =  cm50_store_enum
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  aenm_values_array,
                  store_code_table,
                  status_table,
                  &number_of_values);
```

aenm_values_array—The name of an array of up to 300 fixed-length 8-character arrays (prototyped as cm50$enum300) containing the individual point values to be stored.

18.2.4.5 "C" Call for Point List Store Ordinal Values

```
return_status =  cm50_store_ord
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  oenm_values_array,
                  store_code_table,
                  status_table,
                  &number_of_values);
```

oenm_values_array—The name of an array of up to 300 short integers (prototyped as cm50$int300) containing the individual point values to be stored.

18.2.4.6 "C" Call for Point List Store Internal IDs

```
return_status =  cm50_store_ptid
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  ptid_values_array,
                  store_code_table,
                  status_table,
                  &number_of_values);
```

ptid_values_array—The name of an array of up to 300 64-bit Internal entity ids (prototyped as cm50_ptid_vals) containing the individual point values to be stored.

18.2.4.7 "C" Call for Point List Store Time Values

```
return_status =  cm50_store_time
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  time_values_array,
                  store_code_table,
                  status_table,
                  &number_of_values);
```

time_values_array—The name of an array of up to 300 cm50_time_vals records containing the individual point values to be stored.

18.2.4.8 "C" Call for Point List Store String Values

```
return_status =  cm50_store_stri
                 (&cg_port_num,
                  &priority,
                  acidp_name,
                  point_list_array,
                  stri_values_array,
                  store_code_table,
                  status_table,
                  &number_of_values);
```

stri_values_array—The name of an array of up to 300 fixed-length 40-character arrays (prototyped as cm50_stri_vals) containing the individual point values to be stored.

18.3 SINGLE POINT DATA TRANSFERS

The interface routines in this group Get or Store values from or to one named point.parameter (or parameter array) at a time. For parameter arrays, up to the whole array is accessed. The External ID version of Get Single Point is also used to get LCN date and time.

18.3.1 Single Point Get Data (External ID) Interface

This routine fetches data for a single point from a specified CG or elsewhere on its LCN. The specification of which data is to be fetched and where it is to be stored is contained in the call. For parameter arrays, either a single element, the whole array, or an array subset starting with the first element can be specified. The point may be identified by either a combination of point and parameter names or by a single tag name.

18.3.1.1 Example "C" Calls for Single Point Get

Using point and parameter names as separate variables:

```
return_status =  cm50_get_id
                 (entity,
                  param,
                  &param_ix,
                  &val_loc,
                  &val_st,
                  &val_typ,
                  &cg_port_num);
```

Using a complete tag name:

```
return_status =  cm50_get_tag
                 (tag_name,
                  &val_loc,
                  &val_st,
                  &val_typ,
                  &cg_port_num);
```

18.3.1.2 Parameter Definitions for Single Point Get

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000051 (cm50_lcn_part)—the `val_st` status code for each returned value must be checked.

215000322 (cm50_acc_size)—the array size specified by `param_ix` is larger than the actual size.

tag_name—The name of a string of 40 characters (prototyped as cm50_tag_name_type) that identifies the LCN value(s) to be retrieved. The tag name is formatted as "point.param (param_ix)".

entity—The name of a string of 20 characters (prototyped as cm50_entity_name_type) that contains the ASCII Point ID. It should contain a point name of up to 16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter.

param—The name of a string of 8 characters (prototyped as cm50_ascii_param_arr) that contains the ASCII name of a parameter (or parameter array) from which the value(s) is retrieved.

param_ix—The name of an short integer (prototyped as cm50$int2). Use of this value is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, 5, 13, 15, 17 or 19, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of `param_ix` is used as an index and must be greater than zero. If the parameter being accessed is not an array type, `param_ix` must be zero.

When `val_typ` is 7, 8, 9, 10, 14, 16, 18 or 20, a whole array (or a subset of the array starting with the first element) is to be accessed and `param_ix` is used to specify the number of elements to be accessed—If `param_ix` is smaller than the actual array size, only that number of elements is returned; if it is larger than the actual array size, no elements are returned and the return_status value is 215000322.

val_loc—The name of a program variable where the value(s) are to be stored. The type of variable must match what is declared in `val_typ`.

val_typ	val_loc type
1	float (real)
2	cm50$int2 Integer (short integer)
3	cm50$ascii (fixed-length 24-character array)
4	cm50$enum (fixed-length 8-character array)
5	cm50$int2 Ordinal (short integer)
6	cm50$time_arr Time(fixed-length 18-character array)
7	Array of up to 1000 float
8	Array of up to 1000 cm50$int2 (short Integers)
9	Array of up to 1000 cm50$enum
10	Array of up to 1000 cm50$int2 (Ordinals)
13	cm50_ptid_vals (array of 4 short integers)
14	Array of up to 1000 cm50_ptid_vals
15	cm50_exid_vals (fixed-length 18-character array)
16	Array of up to 1000 cm50_exid_vals
17	cm50_time_vals (record of seconds and ticks)
18	Array of up to 1000 cm50_time_vals
19	cm50_stri_vals (fixed-length 40-character array)
20	Array of up to 1000 cm50_stri_vals

val_st—The name of an short integer (prototyped as cm50$int2) where point-related status information is to be stored. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (215000051). See Appendix A.1 for a listing of Data-Access error/status codes. When `val_typ` specifies an array, `val_st` refers to status of the whole array.

val_typ—The name of an short integer (prototyped as cm50$int2) that contains a number that designates value type of the accessed parameters as listed for **val_loc.**

cg_port_num—The name of an short integer (prototyped as cm50$int2) identifying the CG (1-4) to be accessed.

18.3.2 Single Point Store Data (External ID) Interface

This routine stores data to a single point in a specified CG or elsewhere on its LCN. The specification of where the data is to be found and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

To use this call the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

18.3.2.1 Example "C" Calls for Single Point Store

Using point and parameter names as separate variables:

```
return_status =  cm50_store_id
                 (entity,
                  param,
                  &param_ix,
                  &val_loc,
                  &val_typ,
                  &store_cd,
                  &store_st);
```

Using a complete tag name:

```
return_status =  cm50_store_tag
                 (&tag_name,
                  &val_loc,
                  &val_typ,
                  &store_cd,
                  &store_st);
```

18.3.2.2 Parameter Definitions for Single Point Store

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (cm50_lcn_part), which indicates that the `store_st` status code for each returned value must be checked.

tag_name—The name of a string of 40 characters (prototyped as cm50_tag_name_type) that identifies the LCN value(s) to be stored. The tag name is formatted as "point.param (param_ix)".

entity—The name of a string of 20 characters (prototyped as cm50_entity_name_type) that contains the ASCII Point ID. It should contain a point name of up to 16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter.

param—The name of a string of 8 characters (prototyped as cm50_ascii_param_arr) that contains the ASCII parameter name for the point.parameter where the value is to be stored.

param_ix—The name of an short integer (prototyped as cm50$int2). Use of this value is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, 5, 13, 17, or 19, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of `param_ix` is used as an index and must be greater than zero. If the parameter being accessed is not an array type, `param_ix` must be zero.

When `val_typ` is 7, 8, 9, 10, 14, 18, or 20, a whole array is to be accessed and `param_ix` is used to specify the number of array elements—If `param_ix` does not match the actual array size, no elements are stored and `store_st` indicates an invalid array size.

val_loc—The name of a program variable containing the value(s) to be stored. The type of variable must match what is declared in `val_typ`.

val_typ	val_loc type
1	real number (float)
2	cm50$int2 Integer (short integer)
3	cm50$ascii (fixed-length 24-character array)
4	cm50$enum (fixed-length 8-character array)
5	cm50$int2 Ordinal (short integer)
6	cm50$time_arr Time(fixed-length 18-character array)
7	Array of up to 1000 real numbers (float)
8	Array of up to 1000 cm50$int2 (short Integers)
9	Array of up to 1000 cm50$enum
10	Array of up to 1000 cm50$int2 (Ordinals)
13	cm50_ptid_vals (array of 4 short integers)
14	Array of up to 1000 cm50_ptid_vals
17	cm50_time_vals (record of seconds and ticks)
18	Array of up to 1000 cm50_time_vals
19	cm50_stri_vals (fixed-length 40-character array)
20	Array of up to 1000 cm50_stri_vals

val_typ—The name of an short integer (prototyped as cm50$int2) that contains a number that designates value type of the accessed parameters as listed for **val_loc.**

store_cd—Name of an short integer (prototyped as cm50$int2) that contains a code that allows the substitution of a bad value representation in place of the provided value(s). The store code values are

0 = Store the data value(s) provided
1 = Store the bad value representation instead

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

store_st—The name of an short integer (prototyped as cm50$int2) to contain point-related store status information on completion. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (215000051). See Appendix A.1 for a listing of Data-Access error/status codes. When the `val_typ` is an array, `store_st` refers to status of the whole array.

18.3.3 Single Point Get Data (Internal ID) Interface

This routine fetches data for a single point from the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the CM50_CONV_PT or CM50_CONV_TAG interface, see 19.2.1) reduces the overhead required for repetitive single-point requests.

The specification of which data is to be fetched and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

18.3.3.1 Example "C" Call for Single Point Get

```
return_status =  cm50_getpt_id
                 (id_block,
                  &val_loc),
                  &val_st,
                  &cg_port_num);
```

18.3.3.2 Parameter Definitions for Single Point Get

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (cm50_lcn_part), which indicates that the `val_st` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (prototyped as cm50_idblk) that contains the internal ID data block obtained by a previous Convert External to Internal ID call. When the data is of array type, that call returns the array size in word 7 of the ID block. Thus, if you wish to get less than the entire array you can change the parameter qualifier in the seventh word of the ID block to be smaller than the actual array size. Do not change any other words in the ID block. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

val_loc—The name of a program variable where the value is to be stored. The type of variable must match what was declared in `val_typ` in the earlier Convert ID call.

val_typ	val_loc type
1	real number (float)
2	cm50$int2 Integer (short integer)
3	cm50$ascii (fixed-length 24-character array)
4	cm50$enum (fixed-length 8-character array)
5	cm50$int2 Ordinal (short integer)
6	cm50$time_arr Time(fixed-length 18-character array)
7	Array of up to 1000 real numbers (float)
8	Array of up to 1000 cm50$int2 (short Integers)
9	Array of up to 1000 cm50$enum
10	Array of up to 1000 cm50$int2 (Ordinals)
13	cm50_ptid_vals (array of 4 short integers)
14	Array of up to 1000 cm50_ptid_vals
15	cm50_exid_vals (fixed-length 18-character array)
16	Array of up to 1000 cm50_exid_vals
17	cm50_time_vals (record of seconds and ticks)
18	Array of up to 1000 cm50_time_vals
19	cm50_stri_vals (fixed-length 40-character array)
20	Array of up to 1000 cm50_stri_vals

val_st—The name of an short integer (prototyped as cm50$int2) where point-related status information is to be stored. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (cm50_lcn_part). When the `val_typ` specifies an array, `val_st` refers to status of the whole array.

cg_port_num—The name of an short integer (prototyped as cm50$int2) identifying the CG (1-4) to be accessed.

18.3.4 Single Point Store Data (Internal ID) Interface

This routine stores data to a single point in the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the CM50_CONV_PT or CM50_CONV_TAG interface, see 19.2.1) reduces the overhead required for repetitive single-point requests.

The specification of where the data is found and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

To use this function the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

18.3.4.1 Example "C" Call for Single Point Store

```
return_status =  cm50_storept_id
                 (id_block,
                  &val_loc,
                  &store_cd,
                  &store_st);
```

18.3.4.2 Parameter Definitions for Single Point Store

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (cm50_lcn_part), which indicates that the `store_st` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (prototyped as cm50_idblk) that contains the internal ID data block obtained by a previous Convert External to Internal ID call. Do not change any words in the ID block. If the array size is changed, the array is not stored and the `return_status` value is 215000051, with a `store_st` that indicates an invalid array size. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

val_loc—The name of a program variable that contains the value to be stored. The type of variable must match what was declared in `val_typ` in the earlier Convert ID call.

val_typ	val_loc type
1	real number (float)
2	cm50$int2 Integer (short integer)
3	cm50$ascii (fixed-length 24-character array)
4	cm50$enum (fixed-length 8-character array)
5	cm50$int2 Ordinal (short integer)
6	cm50$time_arr Time(fixed-length 18-character array)
7	Array of up to 1000 real numbers (float)
8	Array of up to 1000 cm50$int2 (short Integers)
9	Array of up to 1000 cm50$enum
10	Array of up to 1000 cm50$int2 (Ordinals)
13	cm50_ptid_vals (array of 4 short integers)
14	Array of up to 1000 cm50_ptid_vals
17	cm50_time_vals (record of seconds and ticks)
18	Array of up to 1000 cm50_time_vals
19	cm50_stri_vals (fixed-length 40-character array)
20	Array of up to 1000 cm50_stri_vals

store_cd—The name of an short integer (prototyped as cm50$int2) that contains a code that allows the substitution of a bad value representation in place of the provided value(s). The store code values are

0 = Store the data value(s) provided
1 = Store the bad value representation instead

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

store_st—The name of an short integer (prototyped as cm50$int2) where point-related status information is to be stored. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (cm50_lcn_part). When the `val_typ` specifies an array, `store_st` refers to status of the whole array.

18.3.5 Get LCN Clock Value Interface

The current date and time as kept by the LCN, can be obtained in either internal or ASCII format. The internal format is a 4-byte integer count of the number of seconds since January 1, 1979. The ASCII format is MM/DD/YYΔHH:MM:SSΔ (where Δ is used to indicate a space).

18.3.5.1 Example "C" Calls to Get the LCN Clock

Internal Time Format:

```
return_status =    cm50_timnow_lcn
                   (&Integer_Clock,
                    &cg_port_num);
```

ASCII Time Format:

```
return_status =    cm50_timnow_asc
                   (&ASCII_Clock,
                    &cg_port_num);
```

18.3.5.2 Parameter Definitions for Get LCN Clock

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

integer_clock—The name of an integer where the clock value, in seconds, is to be stored.

ASCII_clock—The name of a fixed-length 18-character array (prototyped as cm50$time_arr) where the clock value, formatted as 'MM/DD/YY hh:mm:ss ', is to be stored.

cg_port_num—The name of an short integer (prototyped as cm50$int2) identifying the CG (1-4) to be accessed.

18.4 RAW DATA TRANSFERS

The interface routines in this group get, store, and convert **only** LCN **Real data arrays** in LCN format. Each request works only with a single data point's parameter array. These functions allow you to pass Real data arrays from one LCN to another without needing to go through the LCN/VAX data conversions.

18.4.1 Get Raw Data Interface

This function fetches data for a single point from the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the Convert External to Internal ID interface, see 19.2) is required.

The specification of which data is to be fetched and where it is to be stored is contained in the call.

18.4.1.1 Example "C" Call for Get Raw Data

```
return_status =  cm50_spgraw
                 (id_block,
                  value_loc,
                  &priority,
                  &value_status,
                  &cg_port_num);
```

18.4.1.2 Parameter Definitions for Get Raw Data

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (cm50_lcn_part), which indicates that the `value_status` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (prototyped as cm50_idblk), that contains the internal ID data block obtained by a previous Convert External to Internal ID request. When the data is of array type, the conversion returns the array size in word 7 of the ID block. Thus, if you wish to get less than the entire array you can change the parameter qualifier in the seventh word of the ID block to be smaller than the actual array size. Do not change any other words in the ID block. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

value_loc—The name of an array of real numbers (declared as float) where the values are to be stored. The id_block should identify the value type as 7 (Real array).

priority—The name of a short integer (prototyped as cm50$uword) that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

value_status—The name of a short integer (prototyped as cm50$int2) where point-related status information is to be stored. This value is meaningful only when the return_status value indicates normal, complete with errors, or array-size error. See Appendix A.1 for a listing of Data Access error/status codes. Since `val_typ` is 7 (a Real array), `value_status` refers to status of the whole array.

cg_port_num—The name of a short integer (prototyped as cm50$uword) identifying the CG (1-4) to be accessed.

18.4.2 Store Raw Data Interface

This function stores data to a single point in the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the Convert External to Internal ID interface, see 19.2) is required.

The specification of where the data is found and where it is to be stored is contained in the call.

To use this function the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

18.4.2.1 Example "C" Call for Store Raw Data

```
return_status =  cm50_spsraw
                 (id_block,
                  value_loc,
                  &priority,
                  &store_code,
                  &value_status,
                  &cg_port_num);
```

18.4.2.2 Parameter Definitions for Store Raw Data

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (cm50_lcn_part), which indicates that the `value_status` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (prototyped as cm50_idblk), that contains the internal ID data block obtained by a previous Convert External to Internal ID call. Do not change any words in the ID block. If the array size is changed, the array is not stored and the return_status value is 5 with a value_status that indicates an invalid array size. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

value_loc—The name of an array of real numbers (declared as float) that contains the value or values (in LCN format) to be stored. The id_block should identify the value type as 7 (Real array).

priority—The name of a short integer (prototyped as cm50$uword) that contains the requested data-access priority:

1 = High priority (provided for control operations)
2 = Low priority (provided for noncontrol operations)

store_code—The name of a short integer (prototyped as cm50$uword) that contains a code that allows the substitution of a bad value representation in place of the provided value(s). The store code values are:

0 = Store the data value(s) provided
1 = Store the bad value representation (NaN) instead

value_status—The name of a short integer (prototyped as cm50$int2) where point-related status information is to be stored. This value is meaningful only when the return_status value indicates normal or complete with errors. See Appendix A.1 for a listing of Data-Access error/status codes. Since the `val_typ` is 7 (a Real array), `value_status` refers to status of the whole array.

cg_port_num—The name of a short integer (prototyped as cm50$uword) identifying the CG (1-4) to be accessed.

18.4.3 Convert Raw Data

This function converts the elements of a Real array from LCN format to VAX format.

18.4.3.1 Example "C" Call for Convert Raw Data

```
return_status =  cm50_spcraw
                 (id_block,
                  raw_val_loc,
                  vax_val_loc,
                  &value_type,
                  &convert_status);
```

18.4.3.2 Parameter Definitions for Convert Raw Data

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (cm50_lcn_part), which indicates that the `convert_status` status code for each returned value must be checked.

id_block—The name of a 16-byte variable (prototyped as cm50_idblk), that contains the internal ID data block obtained by a previous Convert External to Internal ID call. Do not change any words in the ID block. See heading 4.7.8 in the *Computer Gateway User Manual* for ID block details.

raw_val_loc—The name of an array of real numbers (declared as float) that contains previously obtained raw values that are to be converted from LCN format.

vax_val_loc—The name of an array of real numbers (declared as float) to contain the converted values.

value_type—The name of a short integer (prototyped as cm50$uword) value that must =7 (Real array).

convert_status—The name of a short integer (prototyped as cm50$int2) array where the request-completion status for each data array element is to be stored. Value meanings are

0 = Normal return; this element was converted successfully
1 = Unable to convert this element to VAX format
2 = Bad value substitution was done on this element

18.5 HISTORY DATA TRANSFERS

The interface routines in this group get previously stored averages or 1-minute snapshot data from a History Module on the LCN. The data may be requested using a DDT, Internal Data Block or the internal address of a single tag. The History calls provide for concurrent Get History requests by up-to-four application programs. A fifth request is rejected with a queue-full status return.

18.5.1 Selecting Records from the History Module

The History Module uses a specialized set of circular files to hold historized values collected from data points on the LCN. Effective use of the CM50S history functions requires an understanding of data organization on the History Module.

18.5.1.1 Relative and Absolute Time References

The History Module may be searched using either Relative or Absolute time references. Relative references request data based on a number of records offset from the current value. Absolute Time reference request data for all records whose timestamps fall within a specified Date/Time interval.

For **Absolute Time** references, the Begin Date/Time specifies the timestamp of the most recent value to be retrieved and the End Date/Time specifies the timestamp of the oldest value to be retrieved. If a seasonal time change has occurred during a specified Absolute History interval, the number of samples returned can differ from the expected number of samples. For example, if it is desired to obtain a day's worth of hourly averages (24) and a forward time change of one hour has occurred, 23 samples are returned. If the time change is in the backward direction, 25 samples are returned.

Relative requests are based on beginning and ending offsets which are counts of records back from the current time. The direction of search can be either forward (oldest to newest data) or backwards (newest to oldest data); however, a forward search requires at least twice as long to execute. To execute a backward search, set the starting offset value less-than or equal-to the ending offset value. The number of samples returned is calculated as the positive difference between the starting offset and the ending offset plus one. If this difference exceeds 262, the request is truncated at 262 samples. The number of samples returned by a Relative History request is immune to time changes.

Offset values less than one have special meanings. When the starting or ending offset value is zero (i.e., current LCN time) in the case of averages, the first sample returned is the current running average for the period. A starting offset of -1 has special meaning in the cases of snapshots and user averages. In those cases only, LCN time is rounded to the beginning of the last hour. This permits an ACP to be sure of obtaining the last full hour of snapshots or user averages. In calculating the number of samples returned, a -1 is treated as an offset of 0 and its number of samples and direction of search follows those rules. An ending offset of -1 for snapshots and user averages means the search direction is forward and the ending time is on the hour starting "n" units back from current time.

The following table summarizes results of combinations of starting and ending offsets for Relative History requests with numbers of samples returned and reasons for zero sample returns.

History Type	Starting Offset	Ending Offset	Number of Samples	Direction of Search	Partial First Sample for Averages?
any	0	0	1	Backward	yes
any	1	1	1	Backward	no
any	2	3	2	Backward	no
any	3	2	2	Forward	no
any	0	300	262	Backward	yes
0,5	3	-1	4	Forward	no
1 to 4	3	-1	0	Error, end offset invalid	
0,5	-1	3	4	Backward	no
0,5	-1	-3	0	Error, end offset invalid	
1 to 4	-1	-3	0	Error, begin/end offset invalid	

18.5.1.2 Number of Values Retrieved in a Single Call

The number of values that can be obtained from the History Module for each point is limited both by the size of the buffer used to transfer the values and by the History type. The maximum number of values for monthly averages is 12, and for shift averages is 21. The maximum for user averages is configuration dependent, but will not exceed the number of values shown below for hourly averages. The other maximums are shown in the following table.

Number of Points in DDT or List	Maximum Snapshots	Maximum Hourly Averages	Maximum Daily Averages
1-3	262	168	31
4	262	149	31
5	238	119	31
6	198	99	31
7	170	85	31
8	149	74	31
9	132	66	31
10	119	59	31
11	108	54	31
12	99	49	31
13	91	45	31
14	85	42	31
15	79	39	31
16	74	37	31
17	69	34	31
18	66	33	31
19	62	31	31
20	59	29	29
21	56	28	28
22	53	26	27
23	51	25	25
24	49	24	24

18.5.2 Get History Snapshots (Relative Time)

These routines are used to fetch history snapshots from the HM, using a relative offset from current LCN time.

18.5.2.1 Example "C" Calls for Get History Snapshots (Relative Time)

for standard 1-minute snapshots:

```
return_status =  cm50_ddthis_snap
                 (ddt_name,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  &begin_offset,
                  &end_offset);
```

for fast (5, 10 or 20 second) snapshots:

```
return_status =  cm50_ddthis_fast
                 (ddt_name,
                  &sample_rate,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  &begin_offset,
                  &end_offset);
```

for Multi-Point Lists (instead of DDT):

```
return_status =  cm50_mpl_snap
                 (&mpl_name,
                  &sample_rate,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  &begin_offset,
                  &end_offset,
                  &cg_port_num);
```

for a single data point.parameter:

```
return_status =  cm50_pthis_snap
                 (id_block,
                  &sample_rate,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  &begin_offset,
                  &end_offset,
                  &cg_port_num);
```

18.5.2.2 Parameter Definitions for Get History Snapshots (Relative Time)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (cm50_his_part), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (prototyped as cm50_ddt_name_type) that contains the ASCII name of the DDT to be used.

mpl_name—The name of an Multi-Point List structure defining the data to be retrieved. This should be declared as type cm50_idb_rec.

id_block—The name of a 16-byte variable containing the internal ID for an LCN tag. (prototyped as cm50_idblk). This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

sample_rate —The name of a short integer (prototyped as cm50_uword) identifying the number of snapshots to be returned for each minute. This value does not have to match the rate at which snapshots are historized. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

Note that retrieval of more than 1 snapshot per minute is only supported by LCN release 400 or later.

number_of_values—The name of a short integer (prototyped as cm50_uword) that specifies the maximum number of history items (1..262) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between `begin_offset` and `end_offset`, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with status_table entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points will be lost if the `number_of_values` times the number of points is greater than 1197.

real_values_array—The name of an array of up to 1197 reals (float) where the history data is to be stored. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT.)

status_table—The name of an array of up to 1197 short integers (prototyped as cm50$int2) to contain the value status for each returned snapshot. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: value returned is analog (real) data
1 = Nonstandard: not applicable
2 = Digital Value: value returned is the Real equivalent of an ordinal value for a self-defined enumeration
3-4 = not used
5 = Time Change: a time change occurred and data for one minute is missing; value field contains NaN
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: History collection was not enabled; value field contains NaN
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: not applicable
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX
cm50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
cm50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
cm50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
cm50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
cm50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an array of up to 1197 integers that will contain the time stamp in seconds for each returned snapshot. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) See heading 19.3.3 for time-stamp conversion from internal LCN format to external format.

begin_offset—The name of a short integer (prototyped as cm50$int2) that indicates a relative offset in minutes from current LCN time that represents the starting period for which history is to be fetched.

end_offset—The name of a short integer (prototyped as cm50$int2) that indicates a relative offset in minutes from the current LCN time representing the ending period for which history is to be fetched.

cg_port_num—The name of a short integer (prototyped as cm50$int2) identifying the CG (1-4) to be accessed.

18.5.3 Get History Snapshots (Absolute Times)

These routines are used to fetch history snapshots from the HM, using absolute begin and end times. Separate calls are provided for snapshot and averages histories.

If a seasonal time change has occurred during a specified Absolute History interval, the number of samples returned can differ from the expected number of samples. For example, if it is desired to obtain a day's worth of hourly averages (24) and a forward time change of one hour has occurred, 23 samples are returned. If the time change is in the backward direction, 25 samples are returned.

18.5.3.1 Example "C" call for Get History Snapshots (Absolute Times)

for standard 1-minute snapshots:

```
return_status =  cm50_ddthis_snapt
                 (ddt_name,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_date_time,
                  end_date_time );
```

for fast (5, 10 or 20 second) snapshots:

```
return_status =  cm50_ddthis_fastt
                 (ddt_name,
                  &sample_rate,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_date_time,
                  end_date_time );
```

for Multi-Point Lists (instead of DDT):

```
return_status =  cm50_mpl_snapt
                 (&mpl_name,
                  &sample_rate,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_date_time,
                  end_date_time,
                  &cg_port_num);
```

for a single data point.parameter:

```
return_status =  cm50_pthis_snapt
                 (id_block,
                  &sample_rate,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  begin_date_time,
                  end_date_time,
                  &cg_port_num);
```

18.5.3.2 Parameter Definitions for Get History Snapshots (Absolute Times)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (cm50_his_part), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (prototyped as cm50_ddt_name_type) that contains the ASCII name of the DDT to be used.

mpl_name—The name of an Multi-Point List structure defining the data to be retrieved. This should be declared as type: cm50_idb_rec.

id_block—The name of a 16-byte variable containing the internal ID for an LCN tag. (prototyped as cm50_idblk). This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

sample_rate —The name of a short integer (prototyped as cm50_uword) identifying the number of snapshots to be returned for each minute. This value does not have to match the rate at which snapshots are historized. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

Note that retrieval of more than 1 snapshot per minute is only supported by LCN release 400 or later.

number_of_values—The name of a short integer (prototyped as cm50_uword) that specifies the maximum number of history items (1..261) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between begin and end times, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with `status_table` entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some points are lost if the `number_of_values` times
(1 + the number of points) is greater than 1197.

real_values_array—The name of an array of up to 1197 real numbers (float) where the history data is to be stored. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT).

status_table—The name of an array of up to 1197 short integers (prototyped as cm50$int2) to contain the value status for each returned snapshot. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: value returned is analog (real) data
1 = Nonstandard: not applicable
2 = Digital Value: value returned is the Real equivalent of an ordinal value for a self-defined enumeration
3-4 = not used
5 = Time Change: a time change occurred and data for one minute is missing; value field contains NaN
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: History collection was not enabled; value field contains NaN
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: not applicable
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX
cm50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
cm50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
cm50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
cm50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
cm50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an array of up to 1197 integers that will contain the time stamp in seconds for each returned snapshot. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) See heading 19.3.3 for time-stamp conversion from internal LCN format to external format.

begin_date_time—The name of a 14-character string (prototyped as cm50_lcn_asctim_type) in the format MM/DD/YYΔHH:MM (where Δ indicates a blank character) specifying the date and time for the most recent record to be fetched from the History Module.

The value entered for this parameter should be some time after the desired time period begins, but before the next time period's collection time. For example, begin_date_time for an 11:00 hourly average should be set to any time from 11:01 to 11:59.

end_date_time—The name of a 14-character string (prototyped as cm50_lcn_asctim_type) in the format MM/DD/YYΔHH:MM, specifying the date and time for the oldest record to be fetched from the History Module. The end_date_time must be earlier than begin_date_time.

The value entered for this parameter should be some time after the previous period's collection, but before the desired time period begins. For example, to get an hourly average at 11:00, end_date_time should be set any time between 10:01 and 10:59.

cg_port_num—The name of a short integer (prototyped as cm50_uword) identifying the CG (1-4) to be accessed.

18.5.4 Get History Averages (Relative Times)

These calls return the average, minimum and maximum values for specified time periods.

18.5.4.1 Example "C" call for Get History Averages (Relative Times)

```
return_status =  cm50_ddthis_aver
                 (ddt_name,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  &begin_offset,
                  &end_offset,
                  &history_type);
```

for Multi-Point Lists (instead of DDT):

```
return_status =  cm50_mplhis_aver
                 (&mpl_name,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  &begin_offset,
                  &end_offset,
                  &history_type,
                  &cg_port_num);
```

for a single data point.parameter:

```
return_status =  cm50_pthis_aver
                 (id_block,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  &begin_offset,
                  &end_offset,
                  &history_type,
                  &cg_port_num);
```

18.5.4.2 Parameter Definitions for Get History Averages (Relative Times)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (cm50_his_part), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (prototyped as cm50_ddt_name_type) that contains the ASCII name of the DDT to be used.

mpl_name—The name of an Multi-Point List structure defining the data to be retrieved. This should be declared as type cm50_idb_rec.

id_block—The name of a 16-byte variable (prototyped as cm50_idblk) containing the internal ID for an LCN tag. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of a short integer (prototyped as cm50_uword) that specifies the maximum number of history items (1..262) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between `begin_offset` and `end_offset`, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with `status_table` entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times the number of points is greater than 598.

real_values_array—The name of an array of up to 598 real numbers (float) where the history data is to be stored. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT).

status_table—The name of an array of up to 598 short integers (prototyped as cm50$int2) to contain the value status for each returned snapshot. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX:
cm50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
cm50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
cm50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
cm50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
cm50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an array of up to 598 integers that will contain the time stamp in seconds for each returned average. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) See heading 19.3.3 for time-stamp conversion from internal LCN format to external format.

max_array—The name of an array of up to 598 real numbers (float) that will contain the maximum process value recorded in the averaged period. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT.) Due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of an array of up to 598 real numbers (float) that will contain the minimum process value recorded in the averaged period. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT.) Due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an array of up to 598 unsigned short integers (prototyped as cm50_uword) that will contain the number of samples used in calculating each returned average value. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.)

begin_offset—The name of a short integer (prototyped as cm50$int2) that indicates a relative offset from current LCN time that represents the first history record to be fetched.

end_offset—The name of a short integer (prototyped as cm50$int2) that indicates a relative offset from the current LCN time representing the last history record to be fetched.

history_type—The name of a short integer (prototyped as cm50_uword) that contains the number specifying the type of average requested. The available types and maximum number of records on the History Module for each are:

1 = Hourly	(168	records)
2 = Shift	(21	records)
3 = Daily	(31	records)
4 = Monthly	(12	records)
5 = User	(configuration dependent)	

cg_port_num—The name of a short integer (prototyped as cm50$int2) identifying the CG (1-4) to be accessed.

18.5.5 Get History Averages (Absolute Times)

These calls return the average, minimum and maximum values of a point for specified time periods.

18.5.5.1 Example "C" call for Get History Averages (Absolute Times)

```
return_status =  cm50_ddthis_avert
                 (ddt_name,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time,
                  &history_type);
```

for Multi-Point Lists (instead of DDT):

```
return_status =  cm50_mplhis_avert
                 (&mpl,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time,
                  &history_type,
                  &cg_port_num);
```

for single point requests:

```
return_status =  cm50_pthis_avert
                 (id_block,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time,
                  &history_type,
                  &cg_port_num);
```

18.5.5.2 Parameter Definitions for Get History Averages (Absolute Times)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (cm50_his_part), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (prototyped as cm50_ddt_name_type) that contains the ASCII name of the DDT to be used.

mpl —The name of an Multi-Point List structure defining the data to be retrieved. This should be declared type cm50_idb_rec.

id_block—The name of a 16-byte variable (prototyped as cm50_idblk) containing the internal ID for an LCN tag. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of a short integer (prototyped as cm50_uword) that specifies the maximum number of history items (1..261) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between begin and end times, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with status_table entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times (1 + the number of points) is greater than 598.

real_values_array—The name of an array of up to 598 real numbers (float) where the history data is to be stored. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT.)

status_table—The name of an array of up to 598 short integers (prototyped as cm50$int2) to contain the value status for each returned snapshot. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX

cm50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38

cm50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38

cm50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38

cm50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38

cm50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an array of up to 598 integers to receive the time stamp in seconds for each returned average. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) See heading 19.3.3 for time-stamp conversion from internal LCN format to external format.

max_array—The name of an array of up to 598 real numbers (float) that will contain the maximum process value recorded in the averaged period. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT.) Note that due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of an array of up to 598 real numbers (float) that will contain the minimum process value recorded in the averaged period. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT.) Note that due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an array of up to 598 unsigned short integers (prototyped as cm50_uword) that will contain the number of samples used in calculating each returned average value. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.)

begin_date_time—The name of a 14-character string (prototyped as cm50_lcn_asctim_type) in the format MM/DD/YYΔHH:MM (where Δ indicates a blank character) specifying the date and time for the most recent record to be fetched from the History Module.

The value entered for this parameter should be some time after the desired time period begins, but before the next time period's collection time. For example, begin_date_time for an 11:00 hourly average should be set to any time from 11:01 to 11:59.

end_date_time—The name of a 14-character string (prototyped as cm50_lcn_asctim_type) in the format MM/DD/YYΔHH:MM, specifying the date and time for the oldest record to be fetched from the History Module. The end_date_time must be earlier than the begin_date_time.

The value entered for this parameter should be some time after the previous period's collection, but before the desired time period begins. For example, to get an hourly average at 11:00, end_date_time should be set any time between 10:01 and 10:59.

history_type—The name of a short integer (prototyped as cm50_uword) that contains the number specifying the type of average requested. The available types and maximum time retained on the History Module for each are:

1 = Hourly (7 days)
2 = Shift (7 days)
3 = Daily (31 days)
4 = Monthly (1 year)
5 = User (8 hours to 7 days, depending on configuration)

cg_port_num—The name of a short integer (prototyped as cm50$int2) identifying the CG (1-4) to be accessed.

18.5.6 Get Monthly Averages (Relative Times)

When a point is historized more often than once per minute, it is possible for the number of samples taken during a month to exceed the capacity of a 16-bit integer. This call provides a 32-bit integer count of the number of samples in a monthly average using relative time.

Note: Retrieval of monthly averages using this call is only supported by LCN release 400 and later.

18.5.6.1 Example "C" call for Get Monthly Averages (Relative Times)

```
return_status =  cm50_ddthis_mnth
                 (ddt_name,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  &begin_offset,
                  &end_offset);
```

for Multi-Point Lists (instead of DDT):

```
return_status =  cm50_mplhis_mnth
                 (&mpl_name,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  &begin_offset,
                  &end_offset,
                  &cg_port_num);
```

for a single data point.parameter:

```
return_status =  cm50_pthis_mnth
                 (id_block,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  &begin_offset,
                  &end_offset,
                  &cg_port_num);
```

18.5.6.2 Parameter Definitions for Get Monthly Averages (Relative Times)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (cm50_his_part), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (prototyped as cm50_ddt_name_type) that contains the ASCII name of the DDT to be used.

mpl_name—The name of an Multi-Point List structure defining the data to be retrieved. This should be declared as type cm50_idb_rec.

id_block—The name of a 16-byte variable (prototyped as cm50_idblk) containing the internal ID for an LCN tag. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of a short integer (prototyped as cm50_uword) that specifies the maximum number of history items (1..262) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between `begin_offset` and `end_offset`, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with `status_table` entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times the number of points is greater than 598.

real_values_array—The name of an array of up to 598 real numbers (float) where the history data is to be stored. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT).

status_table—The name of an array of up to 598 short integers (prototyped as cm50$int2) to contain the value status for each returned snapshot. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX:
cm50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38
cm50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38
cm50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38
cm50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38
cm50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an array of up to 598 integers that will contain the time stamp in seconds for each returned average. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) See heading 19.3.3 for time-stamp conversion from internal LCN format to external format.

max_array—The name of an array of up to 598 real numbers (float) that will contain the maximum process value recorded in the averaged period. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT.) Due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of an array of up to 598 real numbers (float) that will contain the minimum process value recorded in the averaged period. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT.) Due to the data compression algorithm on the History Module, there can be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an array of up to 598 integers that will contain the number of samples used in calculating each returned average value. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.)

begin_offset—The name of a short integer (prototyped as cm50$int2) that indicates a relative offset from current LCN time that represents the first history record to be fetched.

end_offset—The name of a short integer (prototyped as cm50$int2) that indicates a relative offset from the current LCN time representing the last history record to be fetched.

cg_port_num—The name of a short integer (prototyped as cm50$int2) identifying the CG (1-4) to be accessed.

18.5.7 Get Monthly Averages (Absolute Times)

When a point is historized more often than once per minute, it is possible for the number of samples taken during a month to exceed the capacity of a 16-bit integer. This call provides a 32-bit integer count of the number of samples in a monthly average using absolute time.

Note: Retrieval of monthly averages using this call is only supported by LCN release 400.

18.5.7.1 Example "C" call for Get Monthly Averages (Absolute Times)

```
return_status =  cm50_ddthis_mntht
                 (ddt_name,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time);
```

for Multi-Point Lists (instead of DDT):

```
return_status =  cm50_mplhis_mntht
                 (&mpl,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time,
                  &cg_port_num);
```

for single point requests:

```
return_status =  cm50_pthis_mntht
                 (id_block,
                  &number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  begin_date_time,
                  end_date_time,
                  &cg_port_num);
```

18.5.7.2 Parameter Definitions for Get Monthly Averages (Absolute Times)

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (cm50_his_part), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (prototyped as cm50_ddt_name_type) that contains the ASCII name of the DDT to be used.

mpl —The name of an Multi-Point List structure defining the data to be retrieved. This should be declared type cm50_idb_rec.

id_block—The name of a 16-byte variable (prototyped as cm50_idblk) containing the internal ID for an LCN tag. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

number_of_values—The name of a short integer (prototyped as cm50_uword) that specifies the maximum number of history items (1..261) to be returned for each point.parameter included in the DDT. If this value is smaller than the actual number of samples found between begin and end times, the number of samples gathered are truncated at this value. If the `number_of_values` is greater than the number of samples returned by the History Module, then the returned arrays are padded with status_table entries of 99 to match the requested `number_of_values`. For multi-point retrievals, values for some of the points are lost if the `number_of_values` times (1 + the number of points) is greater than 598.

real_values_array—The name of an array of up to 598 real numbers (float) where the history data is to be stored. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT.)

status_table—The name of an array of up to 598 short integers (prototyped as cm50$int2) to contain the value status for each returned snapshot. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) If the return_status is CM50_HIS_PART (complete with errors) then for any point that could not be accessed, the first status_table entry will be the Data Access error code (Appendix A.1) for that point. Otherwise, each status_table entry is one of the following value status codes for the corresponding real_values_array entry:

0 = Normal Data: 90% or more good samples
1 = Nonstandard: less than 90% good samples
2 = Digital Value: not applicable (If an average is requested for a parameter of type digital, the value type returned is 12.)
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.
99 = No value (used when fewer than `number_of_values` are returned)

For Floating point values that cannot be represented on the VAX

cm50_Negative_Overflow (16384) = Extremely low value has been clamped to 1.70e-38

cm50_Positive_Overflow (16385) = Extremely high value has been clamped to 1.70e+38

cm50_Negative_Infinity (16386) = IEEE negative infinity value has been clamped to 1.70e-38

cm50_Positive_Infinity (16387) = IEEE positive infinity value has been clamped to 1.70e+38

cm50_NaN (16388) = Bad Value returned as a legitimate (custom data segment) value.

lcn_time_stamp_array—The name of an array of up to 598 integers to receive the time stamp in seconds for each returned average. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.) See heading 19.3.3 for time-stamp conversion from internal LCN format to external format.

max_array—The name of an array of up to 598 real numbers (float) that will contain the maximum process value recorded in the averaged period. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT.) Note that due to the data compression algorithm on the History module, there can be a rounding error of no more than 1% in the reported maximum value for a point.

min_array—The name of an array of up to 598 real numbers (float) that will contain the minimum process value recorded in the averaged period. (The array must be dimensioned for at least `number_of_values` times the number of points listed in the DDT.) Note that due to the data compression algorithm on the History Module, there can be a rounding error of no more than 1% in the reported minimum value for a point.

num_samples_array—The name of an array of up to 598 integers that will contain the number of samples used in calculating each returned average value. (The array must be dimensioned for at least `number_of_values` times the number of points specified in the DDT.)

begin_date_time—The name of a 14-character string (prototyped as cm50_lcn_asctim_type) in the format MM/DD/YYΔHH:MM (where Δ indicates a blank character) specifying the date and time for the most recent record to be fetched from the History Module.

The value entered for this parameter should be some time after the desired time period begins, but before the next time period's collection time. For example, begin_date_time for an 11:00 hourly average should be set to any time from 11:01 to 11:59.

end_date_time—The name of a 14-character string (prototyped as cm50_lcn_asctim_type) in the format MM/DD/YYΔHH:MM, specifying the date and time for the oldest record to be fetched from the History Module. The end_date_time must be earlier than the begin_date_time.

The value entered for this parameter should be some time after the previous period's collection, but before the desired time period begins. For example, to get an hourly average at 11:00, end_date_time should be set any time between 10:01 and 10:59.

cg_port_num—The name of a short integer (prototyped as cm50$int2) identifying the CG (1-4) to be accessed.

18.5.8 Historization Sampling Rate Queries

These functions query the LCN and return the current Historization Sampling Rate (number of snapshots recorded each minute) for a point or set of points.

Note: Retrieval of sampling rates using this call is only supported by LCN release 400 or later.

18.5.8.1 Example "C" calls for Query Sampling Rate

For Points referenced in a History DDT:

```
return_status =  cm50_ddthis_rate
                 (ddt_name,
                  history_rate_array,
                  status_table);
```

For a List of Internal Point ids:

```
return_status =  cm50_mplhis_rate
                 (mpl_name,
                  history_rate_array,
                  status_table,
                  &cg_port_number);
```

For a Point addressed by its internal id:

```
return_status =  cm50_pthis_rate
                 (id_block,
                  &history_rate,
                  &cg_port_number);
```

For a Point addressed by its internal id:

```
return_status =  cm50_taghis_rate
                 (tagname,
                  &history_rate,
                  &cg_port_number);
```

18.5.8.2 Parameter Definitions for History Sampling Rate Queries

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000651 (cm50_his_part), which indicates that the `status_table` status code for each returned value must be checked.

ddt_name—The name of a 9-character string (prototyped as cm50_ddt_name_type) that contains the ASCII name of the DDT to be used.

mpl —The name of an Multi-Point List structure defining the data to be retrieved. This should be declared type cm50_idb_rec.

id_block—The name of a 16-byte variable (prototyped as cm50_idblk) containing the internal ID for an LCN tag. This value will have been obtained through a previous Convert External to Internal ID call. Note: Array elements must be specified individually; this argument cannot be used to obtain history for an entire array.

tagname—The name of a 40-character string (prototyped as cm50_tagname_type) that contains an LCN tagname in the form "point.parameter(index)", where the "(index)" is used only to identify elements of an array.

history_rate —The name of a short integer (prototyped as cm50$int2) identifying the number of snapshots collected each minute. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

history_rate_array —The name of an array of short integers (prototyped as cm50$hist_array) identifying the number of snapshots collected each minute. Acceptable values are:

1 for 1-minute snapshots
3 for 20-second snapshots
6 for 10-second snapshots
12 for 5-second snapshots.

status_table—The name of an array of short integers (declared as cm50$hist_array) to contain the data access code for each point (See appendix A.1).

18.6 TEXT MESSAGE TRANSFERS

The two interface routines in this group are used to send and receive character-string messages over the LCN.

18.6.1 Get Message Interface

This routine is used to fetch a character-string message held in a buffer by this program's ACIDP. The message presence is determined as the result of a Get ACP Status request.

18.6.1.1 Example "C" Call for Get Message

```
return_status =  cm50_getmsg
                 (msg,
                  &msg_len);
```

18.6.1.2 Parameter Definitions for Get Message

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. There are three non-normal return_status values for this call that indicate the need for additional processing:

215000521	CM50_MSG_TRUNC	Received message was truncated
215000561	CM50_MSG_QUE	Message was received and another one is queued
215000571	CM50_MSG_QUET	Received message was truncated & another one is queued

msg—The name of a fixed-length 120-character array (prototyped as cm50$msg_string) where the message is to be stored.

msg_len—The name of a short integer (prototyped as cm50_uword) that specifies the maximum number of characters to accept (1 to 120).

18.6.2 Send Message Interface

This routine is used to send a message to all operator stations assigned to the same unit as this program's ACIDP. A request to wait for operator confirmation is optional. If operator confirmation is requested, execution of the requesting program is suspended until either the confirmation occurs, or until its specified wait time expires. The requesting program receives an indication of whether confirmation or a time out occurs.

18.6.2.1 Example "C" Call for Send Message

```
return_status =  cm50_storemsg
                 (msg,
                  &msg_len,
                  &confirm,
                  &timeout,
                  &dest);
```

18.6.2.2 Parameter Definitions for Send Message

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

msg—The name of an array of up to 120 characters (not terminated by a null character) that contains the message to be sent.

msg_len—The name of a short integer (prototyped as cm50_uword) that specifies the number of characters to be transmitted. The maximum number of characters depends on message destination: 60 for CRT displays and 72 for printing. Over-length messages are truncated. All messages are archived if the HM is so configured.

confirm—The name of a Boolean short integer (prototyped as cm50$bool2 with 1 = TRUE and 0 = FALSE) that specifies whether or not a message confirmation is required. Note that this parameter is treated as FALSE if the message destination is printer only.

timeout—The name of a short integer (prototyped as cm50_uword) that specifies the number of seconds (0 to 3600) the system is to wait for confirmation before returning control to the requesting program with a "no confirm" return_status. (Allow for a built-in time lag of up-to-10 seconds.) The Wait Time parameter is ignored if the Confirm parameter is set to OFF or the message destination is printer only.

dest—The name of a short integer (prototyped as cm50$int2) that specifies where the message is to be sent, as follows:

0 – CRT only
1 – Printer only
2 – Both

18.6.2.3 Event-Initiated Reports

Two types of Event-Initiated Reports can be invoked by specially formatted messages from an ACP or an Indirect Control Program to the Area Universal Stations:

- Logs, reports journals, and trends configured in the Area Database
- Event History reports

Details of message requirements are given in Section 30 of the *Engineer's Reference Manual* located in the *Implementation/Startup & Reconfiguration - 2* binder.

PROGRAM CONTROL AND SUPPORT ("C")
Section 19

This section discusses program interfaces that control the execution of ACPs and convert values between formats used in the VAX and on the TDC 3000 Local Control Network.

19.1 ACP EXECUTION SUPPORT

These interface routines affect the orderly execution and termination of application programs.

19.1.1 ACP Initialization Interface

This routine (or the vintage ACPTRP procedure) **must** be the first executable statement in each ACP but is optional for DAPs and Indirect Control Programs. It establishes a termination handler and ensures proper ACP table setup. Failure to invoke this interface routine as the first statement of an ACP may not appear to cause immediate problems, but will result in improper termination handling. The termination status is not reported to the CG, and the ACP appears to both the CM50S and the CG to still be in the RUN state even though the process has terminated.

The call to CM50_SET_ACP also establishes a system lock that allows the program to be terminated cleanly if CM50S is shut down. Therefore, it is advisable to include this call in every program that is mapped to the CM50S shareable image.

19.1.1.1 Example "C" Call for ACP Initialization

```
return_status = cm50_set_acp
                        (&reset);
```

19.1.1.2 Parameter Definitions for ACP Initialization

return_status—The name of an integer to receive the overall return status of the function. This function always returns as a success (`return_status` = 1).

reset—The name of an INTEGER*2 that specifies the reaction of the trap handler to an abort. If the ACP is aborted for any reason, the Abort code is recorded in the CG/PLNM database and the ACP Status table. If the value of **reset** is 1, then the execution status of the ACP is reset to OFF/DELAY regardless of how the program terminated. For any other value of **reset**, the execution status of the ACP becomes OFF/DELAY only after normal termination and is set to ABORT after an abnormal program termination.

19.1.2 Get ACP Status Interface

This routine fetches a set of parameters that enables the requesting ACP to determine why the system has turned it on and what special processing may be required at this time. It should be used during both the "setup" and "cleanup" program stages each time an ACP runs. After servicing this request, the interface routine resets its copy of these values in preparation for any subsequent ACP turn on.

NOTE

GETSTS is one of the few CM50S user-interface routines that is not implemented as a function. It is called as a "C" procedure.

19.1.2.1 Example "C" Call for Get ACP Status

```
getsts ( &take_i_p,
         &ps_msg,
         &demand,
         &procspec,
         &scheduled,
         &upper_level);
```

19.1.2.2 Parameter Definitions for Get ACP Status

take_i_p—The name of a Boolean short integer (prototyped as cm50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE the first time this program is turned on by the CG, following an initialization event (see heading 4.4.1). `take_i_p` should be ignored when `upper_level` is TRUE.

ps_msg—The name of a Boolean short integer (prototyped as cm50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE if a message for the program is waiting at the CG.

demand—The name of a Boolean short integer (prototyped as cm50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on as the result of a process operator request.

procspec—The name of a Boolean short integer (prototyped as cm50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on as the result of a process special to its ACIDP from an HG, AM, or another ACP.

scheduled—The name of a Boolean short integer (prototyped as cm50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on by periodic or cyclic scheduling.

upper_level—The name of a Boolean short integer (prototyped as cm50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE if the program was turned on by the VAX.

19.1.3 ACP Delay Interface

This routine suspends execution of the calling program for a specified number of seconds. Program execution resumes at the statement following the delay call.

19.1.3.1 Example "C" Call for ACP Delay

```
sleep = cm50_acpdelay
        (&delay_time);
```

19.1.3.2 Parameter Definitions for ACP Delay

sleep—The name of a Boolean short integer (prototyped as cm50$bool2 with 1 = TRUE and 0 = FALSE) that contains the overall return status of the function call. The value will be FALSE when the call has been rejected because of an invalid delay time value.

delay_time—The name of a short integer (prototyped as cm50_uword) that contains the length of time (1 to 60 seconds) that the requesting program is to be suspended.

19.1.4 ACP Hibernate Interface

This routine suspends execution of the calling ACP (through a VMS SYS$HIBER request) until the next turn on request. The program and associated data remain in memory during hibernation, in effect making it memory-resident. Program execution resumes at the statement following the CM50_HIBER call.

19.1.4.1 Example "C" Call for ACP Hibernate

```
hiber_stat = cm50_hiber ();
```

Note that the empty argument list "( )" is required when calling the function from "C".

19.1.4.2 Parameter Definitions for ACP Hibernate

hiber_stat—The name of an integer to receive the overall return status of the function call. This value should always = 1 (SS$_NORMAL). Any other value indicates a fatal error. The program should call the GETSTS routine (see heading 19.1.2) to determine how the Wake call was issued.

19.1.5 ACP Termination Interface

This routine terminates the execution of the calling ACP. It must be used as the last operating statement of each ACP but is optional for DAPs and Indirect Control Programs.

For ACPs, this call stores a termination-status code in the associated ACIDP's ABORTCOD parameter. The termination code can be viewed at a Universal Station (see the definitions for ABORTCOD and EXECSTAT at heading 4.4.1), but in a revised form. The integer value assigned here is translated into two hexadecimal digits (00 to FF) and appended to the character string EA. Thus, an ACP-assigned abnormal termination code of 15 appears at the Universal Station display as EA0F.

If an ACP is aborted by the VMS operating system, an abort code of VMSF is stored in its ACIDP's ABORTCOD.

The execution state of an ACIDP can be changed from ABORT to normal by operator demand through a Universal Station or by invoking the ACP Operation screen's Deactivate/Terminate function. See heading 5.8 for abort recovery details.

NOTE

PRGTRM is one of the few user-interface routines that is not implemented as a function. It is called as a "C" procedure.

19.1.5.1 Example "C" Call for ACP Termination

```
PRGTRM (&terminate_code);
```

19.1.5.2 Parameter Definitions for ACP Termination

terminate_code—The name of an integer that must contain zero or a positive value (1 to 255). Zero value indicates normal termination. Nonzero values are user-specified codes for nonnormal termination (abort). Note that if you provide a value outside the valid range, ABORTCOD will contain EAΔΔ (where.Δ represents a blank).

19.2 ENTITY NAME CONVERSIONS

The interface routines in this group convert ASCII references to tags on the LCN to their internal LCN identifiers.

NOTE

The all internal point.parameter addresses need to be rebuilt and the program(s) using them need to be recompiled whenever the LCN database is changed in a significant manner, such as by the rebuild or deletion of data points referenced in the address array.

19.2.1 Convert External to Internal ID

These routines fetch the internal ID of a point.parameter for the calling program. Use of the internal ID by repetitive single-value data gets and stores reduces system overhead and provides faster return of data. The specification of which point.parameter internal ID is wanted and where it is to be stored is contained in the call.

19.2.1.1 Example "C" Calls for Convert ID

Using point and parameter names as separate variables:

```
return_status =  cm50_conv_pt
                      (entity,
                       param,
                       &param_ix,
                       id_block,
                       &val_typ,
                       &cg_port_num);
```

When the external id is expressed as a Tag name (not separate point and parameter), use:

```
return_status =  cm50_conv_tag
                      (tag_name,
                       id_block,
                       &val_typ,
                       &cg_port_num);
```

19.2.1.2 Parameter Definitions for Convert ID

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially the following return codes:

215000146 (cm50_lcn_array)—the array size specified by `param_ix` is smaller the actual array size.

215000322 (cm50_acc_size)—the array size specified by `param_ix` is larger than the actual array size.

tag_name—The name of a 40-character string (prototyped as cm50_tag_name_type) that identifies the LCN value(s) to be stored. The tag name is formatted as "point.param (param_ix)".

entity—The name of a 20-character string (prototyped as cm50_entity_name_type) that contains the ASCII Point ID. It should contain a point name of up to 16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter for Network Gateway routing.

param—The name of an 8-character string (protoyped as cm50_ascii_param_arr) that contains the LCN parameter name.

param_ix—The name of a short integer (prototyped as cm50$int2). Use of this value is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, 5, 13, 15, 17 or 19, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of `param_ix` is used as an index and must be greater than zero. If the parameter being accessed is not an array type, `param_ix` must be zero.

When `val_typ` is 7, 8, 9, 10, 14, 16, 18 or 20, a whole array (or a subset of the array starting with the first element) is to be accessed and `param_ix` is used to specify the number of elements to be accessed. If `param_ix` is smaller than the actual array size, the conversion is made; if it is larger than the actual array size, the conversion is not made. Both conditions cause non-normal `return_status` values to be returned.

id_block—The name of a 16-byte variable (prototyped as cm50_idblk) where the internal ID data block is to be returned. Save these eight values for later use in calls on this point.parameter. The ID data block contents are as follows:

Word 1—	Data type
Words 2..5—	Internal point identifier
Word 6—	Parameter subscript
Word 7—	Parameter qualifier (array size)
Word 8—	Enumeration set identifier

val_typ—The name of a short integer (prototyped as cm50_uword) that contains a number that designates value type. If the incorrect value is supplied on input, this value will be updated as an output variable. The coded values:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
6 = not used
7 = Real array
8 = Integer array
9 = Enumeration array
10 = Ordinal value of enumeration array
13 = Internal entity id
14 = Internal entity id array
15 = External entity id
16 = External entity id array
17 = Time value
18 = Time value array
19 = String value
20 = String value array

cg_port_num—The name of a short integer (prototyped as cm50_uword) identifying the CG (1-4) to be accessed.

19.2.2 Convert List of External IDs

These routines fetch an array of internal IDs for a list of point.parameters. These calls are designed for use with the Point Array calls described in section 9.4. All of the point.parameters in each list must be of the same data type (Real, ASCII, etc.).

19.2.2.1 Example "C" Call for Convert Lists

When the point & parameter names are maintained separately, use:

```
return_status =  cm50_conv_pt_list
                      (entity_arr,
                       param_arr,
                       param_ix_arr,
                       &number_of_values,
                       &val_typ,
                       &cg_port_num
                       id_block_arr,
                       return_arr);
```

When the external id is expressed as a Tag name (not separate point and parameter), use:

```
return_status =     cm50_conv_tag_list
                    (tagname_arr,
                     &number_of_values,
                     &val_typ,
                     &cg_port_num
                     id_block_arr,
                     return_arr);
```

19.2.2.2 Parameter Definitions for Convert Lists

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values. Note especially return code 215000051 (CM50_lcn_part), which indicates that the `return_arr` array entry for each returned id block must be checked for errors.

return_arr—The name of an array of up to 300 longwords to receive the status of the conversion of each point or tag. See Appendix A.2 for an explanation and a listing of all assigned return code values.

tagname_arr—The name of an array of up to 300 40-character strings (prototyped as cm50_tag_list_type). Each string contains the ASCII tagname of an LCN entity for which the internal ID is to be obtained. The tagames are formatted as `Point.Parameter` or `Point.Parameter(ix)`, where (ix) is an array element index used only with array parameters.

entity_arr—The name of an array of up to 300 20-character strings (prototyped as cm50_entity_list_type), each containing an ASCII Point id. It should contain a point name of up-to-16 characters, optionally preceded by a 1- or 2-character pinid and a backslash (\) delimiter for Network Gateway routing.

param_arr—The name of an array of up to 300 8-character strings (prototyped as cm50_param_names_type), each containing the ASCII parameter name of a point.parameter for which the internal ID is to be obtained.

param_ix_arr—The name of an array of up to 300 short integers (prototyped as cm50_intg_array_type) used as array element index values corresponding to the individual parameter names in param_arr. For each non-array parameter named in that array, the corresponding value in this array should be zero.

cg_port_num—The name of a short integer (prototyped as cm50_uword) identifying the CG (1-4) to be accessed.

id_block_arr—The name of an array of up to 300 16-byte variables (prototyped as cm50_idblk) where the internal ID data blocks are to be returned. The ID data block contents are as follows:

Word 1—	Data type
Words 2..5—	Internal point identifier
Word 6—	Parameter subscript
Word 7—	Parameter qualifier (array size)
Word 8—	Enumeration set identifier

number_of_values—The name of a short integer (prototyped as cm50_uword) that contains the number of points/tags in the list to be converted.

val_typ—The name of a short integer (prototyped as cm50_uword) that contains a number that designates LCN value type. This value must be supplied in the calling argument. The acceptable values are:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
13 = Internal entity id
15 = External entity id
17 = Internal Time
19 = String

19.3 VALUE CONVERSIONS

The interface routines in this group convert values between different formats used in the LCN and the host computer.

19.3.1 Valid Number Check

This routine checks a value of type "Real" to determine if it is a valid single-precision, floating-point number. Its primary purpose is to check for the "Bad Value" indicator, NaN (-0).

19.3.1.1 Example "C" Call for Valid Number Check

```
value_st =    cm50_validn (&value);
```

19.3.1.2 Parameter Definitions for Valid Number Check

value_st—The name of a Boolean short integer (prototyped as cm50$bool2 with 1 = TRUE and 0 = FALSE) that returns TRUE if "Value" is found to be a valid floating-point number. It returns FALSE for minus zero (NaN) or other invalid bit configurations.

value—The name of a Real variable (float type) that contains the variable that is to be checked. When `value_st` returns FALSE, the contents of `value` have been changed to 0.0.

19.3.2 Set Bad Value

This routine stores the bad value constant, NaN (-0), into the specified Real variable.

19.3.2.1 Example "C" Call for Set Bad Value

```
return_status = cm50_setbad (&var_name);
```

or

```
cm50_setbad (&var_name);
```

19.3.2.2 Parameter Definitions for Set Bad Value

return_status—The name of an integer to receive the overall return status of the function call. For this function, `return_status` = 1 always.

var_name—The name of a Real variable (float type) where the bad value constant for the LCN is to be stored. Note that attempting to perform arithmetic operations or numeric conversions using a variable that has been set to Bad Value can cause a fatal error within the program.

19.3.3 Convert Time Values

Within the CM50 environment, Date/time variables are often maintained in a variety of formats. The following routines convert time values from any one of the following formats to any other:

abbrev.	format	use
LCN	4-byte integer	internal LCN clock, number of seconds since January 1, 1979
VAXB	8-bytes (array of two 4-byte integers)	VAX binary system clock format
VAXA	22 characters	VAX standard ASCII time display: 'dd-MON-yyyy hh:mm:ss'
ASC	18 characters	LCN standard ASCII time display 'mm/dd/yy hh:mm:ss'
EURO	18 characters	European ASCII time display 'dd/mm/yy hh:mm:ss'
ARY	12 bytes (equivalenced to six short integers)	array of short int with element: 1 = year 2 = month 3 = day 4 = hour 5 = minute 6 = second

In each routine, the first argument must be assigned the input value and the second argument is the returned converted value.

19.3.3.1 Example "C" Calls to Convert Time

Convert internal LCN time to an array of short integers (prototyped as cm50_uword):

```
return_status = cm50_timlcn_ary
                      (&lcn,
                       ary);
```

Convert internal LCN time to an ASCII string:

```
return_status = cm50_timlcn_asc
                      (&lcn,
                       asc);
```

Convert internal LCN time to a European string:

```
return_status = cm50_timlcn_euro
                      (&lcn,
                       euro);
```

Convert internal LCN time to VAX display format:

```
return_status = cm50_timlcn_vaxa
                      (&lcn,
                       vaxa);
```

Convert internal LCN time to VAX binary:

```
return_status = cm50_timlcn_vaxb
                      (&lcn,
                       &vaxb);
```

Convert a short integer array to internal LCN:

```
    return_status = cm50_timary_lcn
                          (ary,
                           &lcn);
```

Convert a short integer array to an ASCII string:

```
    return_status = cm50_timary_asc
                          (ary,
                           asc);
```

Convert a short integer array to a European string:

```
    return_status = cm50_timary_euro
                          (ary,
                           euro);
```

Convert a short integer array to VAX display format:

```
    return_status = cm50_timary_vaxa
                          (ary,
                           vaxa);
```

Convert a short integer array to VAX binary:

```
    return_status = cm50_timary_vaxb
                          (ary,
                           &vaxb);
```

Convert an ASCII string to internal LCN:

```
    return_status = cm50_timasc_lcn
                          (asc,
                           &lcn);
```

Convert an ASCII string to a short integer array:

```
    return_status = cm50_timasc_ary
                          (asc,
                           ary);
```

Convert an ASCII string to a European string:

```
    return_status = cm50_timasc_euro
                          (asc,
                           euro);
```

Convert an ASCII string to VAX display format:

```
    return_status = cm50_timasc_vaxa
                          (asc,
                           vaxa);
```

Convert an ASCII string to VAX binary:

```
    return_status = cm50_timasc_vaxb
                          (asc,
                           &vaxb);
```

Convert a European string to internal LCN:

```
return_status = cm50_timeuro_lcn
                        (euro,
                         &lcn);
```

Convert a European string to a short integer array:

```
return_status = cm50_timeuro_ary
                        (euro,
                         ary);
```

Convert a European string to an ASCII string:

```
return_status = cm50_timeuro_asc
                        (euro,
                         asc);
```

Convert a European string to VAX display format:

```
return_status = cm50_timeuro_vaxa
                        (euro,
                         vaxa);
```

Convert a European string to VAX binary:

```
return_status = cm50_timeuro_vaxb
                        (euro,
                         &vaxb);
```

Convert VAX display format to internal LCN:

```
return_status = cm50_timvaxa_lcn
                        (vaxa,
                         &lcn);
```

Convert VAX display format to a short integer array:

```
return_status = cm50_timvaxa_ary
                        (vaxa,
                         ary);
```

Convert VAX display format to an ASCII string:

```
return_status = cm50_timvaxa_asc
                        (vaxa,
                         asc);
```

Convert VAX display format to a European string:

```
return_status = cm50_timvaxa_euro
                        (vaxa,
                         euro);
```

Convert VAX display format to VAX binary:

```
return_status = cm50_timvaxa_vaxb
                        (vaxa,
                         &vaxb);
```

Convert VAX binary to internal LCN:

```
return_status = cm50_timvaxb_lcn
                        (&vaxb,
                         &lcn);
```

Convert VAX binary to a short integer array:

```
return_status = cm50_timvaxb_ary
                        (&vaxb,
                         ary);
```

Convert VAX binary to an ASCII string:

```
return_status = cm50_timvaxb_asc
                        (&vaxb,
                         asc);
```

Convert VAX binary to a European string:

```
return_status = cm50_timvaxb_euro
                        (&vaxb,
                         euro);
```

Convert VAX binary to VAX display format:

```
return_status = cm50_timvaxb_vaxa
                        (&vaxb,
                         vaxa);
```

19.3.3.2 Parameter Definitions for Convert Time Values

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

lcn—The name of an integer that contains a value representing internal LCN time to the nearest second.

ary—The name of an array of 6 short integers (prototyped as cm50_integer_time_type) that contains a value representing a date and time.

asc—The name of a fixed-length 18-character array (prototyped as cm50$time_arr) representing time in the format : 'mm/dd/yy hh:mm:ss '.

euro—The name of a fixed-length 18-character array (prototyped as cm50$time_arr) representing time in the format : 'dd/mm/yy hh:mm:ss '.

vaxa—a fixed-length 22-character array (prototyped as cm50_lcn_ascii_time_type) representing time in the format: 'dd-MON-yyyy hh:mm:ss', where MON represents the first three letters (in upper case) of the English name of the month.

vaxb—The name of a 64-bit variable (prototyped as cm50_vms_binary_time_type) that contains a value representing internal VAX binary time.

CM50S ADMINISTRATION ("C")
Section 20

This section discusses the programmatic calls that can be used to manage the ACPs and DDTs installed in a CM50S system.

20.1 PROGRAMMATIC INTERFACES TO ACP OPERATIONS

A programmatic interface to all ACP Operations gives users programmatic access to the same ACP functions that are available through the ACP Operations user interface. In order to use the ACP Programmatic Interface, the user should include the ACP Include files (`CM50_FLAGS_INCLUDE.H` and `CM50_ACP_INCLUDE.H`). These files define data types and routines required by the Programmatic Interface calls. The following sections discuss the ACP Programmatic Interface calls in detail.

20.1.1 Install ACP

This routine is called to install an ACP. The ACP can be installed under a different name than the executable filename.

20.1.1.1 Example "C" Call for Install ACP

```
return_status = cm50_acp_install
                (acp_name,
                 process_name,
                 mailbox_name,
                 exe_path,
                 &mode,
                 input_path,
                 output_path,
                 error_path,
                 &privilege,
                 uic,
                 &priority,
                 &creprc_flags,
                 &quota_list,
                 &flags);
```

20.1.1.2 Parameter Definitions for Install ACP

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name (12-character string, prototyped as acp_name_type) of the ACP to be installed. Must be specified.

process_name—The name (15-character string, prototyped as proc_name_type) to be assigned to the created process. If set to spaces, the ACP name will be used. Note: Each process must have a unique name. The activation of an ACP will fail if a process with the specified process_name is active on the system.

mailbox_name—The name (40-character string, prototyped as mbx_log_type) of a Mailbox (normally set to spaces) to receive a termination message when the created process (ACP) is complete. This is a temporary termination mailbox created by the Programmatic Interface and ACPOPER utility. For more information, refer to the VMS System Services Reference Manual. This mailbox parameter is applicable only when the ACP is executed as a remote (detached) process. An ACP run interactively ignores the mailbox parameter in the ACP table. The mailbox is created using VMS defaults.

exe_path—Full pathname (80-character string, prototyped as path_name_type) of the executable file. If set to spaces, the default is the executable file specified by the acp_name in the CM50$ACP directory.

mode— Short integer code (prototyped as acp_mode_type) Specifies what mode to install the ACP in. The values are:

1 = TEST
2 = RESTRICTED
3 = NORMAL

input_path—Full pathname (80-character string, prototyped as path_name_type) of the alternate input filename.

output_path—Full pathname (80-character string, prototyped as path_name_type) of the alternate output filename. If left blank, SYS$OUTPUT will be directed to the NULL device.

error_path—Full pathname (80-character string, prototyped as path_name_type) of the alternate error filename. If left blank, SYS$ERROR will be directed to the NULL device.

privilege—Privileges specification. Declared as type: priv_mask_type, it assigns special VMS privileges to the ACP. Set both components (.L0 and .L1) to zero for a normal, unprivileged ACP.

uic—The name (12-character string, prototyped as path_name_type) of the user whose UIC is to be used when the ACP is executed remotely. Only the first 12 characters are signifcant; the remainder should be blank filled.

priority—Unsigned integer specifying the VMS priority (0-30) of the ACP process.

creprc_flags—Unsigned integer (prototyped as creprc_flag_type) VMS Create Process flags specification. Normally set to zero.

quota_list—Quotas specification. Declare as type : quota_list_type.. The last element of the array must have a **quota_tag** = zero. To use the system defaults, pass a single integer with a value of zero.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

20.1.2 Uninstall ACP

This routine is called to uninstall an ACP.

20.1.2.1 Example "C" Call for Uninstall ACP

```
return_status = cm50_acp_uninst
                (acp_name,
                 &flags);
```

20.1.2.2 Parameter Definitions for Uninstall ACP

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name (12-character string, prototyped as acp_name_type) of the ACP that is to be uninstalled.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

20.1.3 Activate ACP

This routine is called to activate an installed ACP under a mode specified by the user.

20.1.3.1 Example "C" Call for Activate ACP

```
return_status = cm50_acp_act
                (acp_name,
                 &mode,
                 &flags);
```

20.1.3.2 Parameter Definitions for Activate ACP

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name (12-character string, prototyped as acp_name_type) of the ACP to be activated.

mode—This short integer (prototyped as act_mode_type) specifies whether the ACP is to be activated as a REMOTE detached process (mode = 0) or as an INTERACTIVE subprocess (mode = 1).

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

20.1.4 Deactivate ACP

This routine is called to deactivate an installed ACP, placing it in a specified state.

20.1.4.1 Example "C" Call for Deactivate ACP

```
return_status = cm50_acp_deactivate
                (acp_name,
                 &state,
                 &flags);
```

20.1.4.2 Parameter Definitions for Deactivate ACP

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name (12-character string, prototyped as acp_name_type) of the running ACP to be deactivated.

state—This short integer (prototyped as exec_state_type) specifies whether to set the ACIDP to an ABORT (state = 0) or OFF/DELAY (state = 3).

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

20.1.5 Connect ACP to an ACIDP

This routine is called to connect an installed ACP to an ACIDP on the LCN.

20.1.5.1 Example "C" Call for Connect ACP to an ACIDP

```
return_status = cm50_acp_connect
                (acp_name,
                 acidp_name,
                 &cg_port_number,
                 &flags);
```

20.1.5.2 Parameter Definitions for Connect ACP to an ACIDP

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name (12-character string, prototyped as acp_name_type) of the ACP to be connected.

acidp_name—The name (16-character string, prototyped as cm50_long_acidp) of the ACIDP to connect the to the ACP.

cg_port_number—This short integer specifies which CG (1-4) contains the `ACIDP.`

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

20.1.6 Disconnect ACP from an ACIDP

This routine is called to disconnect an installed ACP from an ACIDP on the LCN.

20.1.6.1 Example "C" Call for Disconnect ACP from an ACIDP

```
return_status = cm50_acp_discon
                (acp_name,
                 &flags);
```

20.1.6.2 Parameter Definitions for Disconnect ACP from an ACIDP

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name (12-character string, prototyped as acp_name_type) of the ACP to be disconnected.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

`cm50$m_handler`
`cm50$m_msgon`

20.1.7 Change ACP Mode

This routine is called to change the installation mode of an ACP.

20.1.7.1 Example "C" Call for Change ACP Mode

```
return_status = cm50_acp_chg_mode
                 (acp_name,
                  mode,
                  flags);
```

20.1.7.2 Parameter Definitions for Change ACP Mode

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name (12-character string, prototyped as acp_name_type) of the ACP whose mode is to be changed.

mode—This short integer (prototyped as act_mode_type) specifies the new mode of the ACP. Permitted values are:

1 = TEST
2 = RESTRICTED
3 = NORMAL

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

`cm50$m_handler`
`cm50$m_msgon`

20.1.8 Get ACP Summary

This routine is called to get summary information for an installed ACP. The output optionally can be sent to the printer.

20.1.8.1 Example "C" Call for Get ACP Summary

```
return_status = cm50_acp_sum
                (acp_name,
                 &acp_summary,
                 &flags);
```

20.1.8.2 Parameter Definitions for Get ACP Summary

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The name (12-character string, prototyped as acp_name_type) of the ACP for which summary information is to be returned.

acp_summary—This argument (declared as type: acp_summary_rec) specifies where the summary information is to be returned.

Note that the acp_summary_rec structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

`cm50$m_handler`
`cm50$m_msgon`

20.1.9 Get List of ACPs

This routine is called to get a list of installed ACPs. Up to 400 ACPs will be reported in a single call, if more than 400 ACPs are installed on a system, repeating the call with a start_rec of 1, 401 and 801 will return additional ACPs until the system maximum of 1000 have been returned.

20.1.9.1 Example "C" Call for Get List of ACPs

```
RETURN_STATUS = cm50_acp_listall
                (&start_rec,
                 &end_rec,
                 &total_returned,
                 &list,
                 &flags);
```

20.1.9.2 Parameter Definitions for Get List of ACPs

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

start_rec—An integer specifying the starting (lowest) record number within the ACP status table to be reported.

end_rec—An integer specifying the ending (highest) record number within the ACP status table to be reported.

total_returned—An integer value specifying the number of records actually returned. This may be less than the number requested if the end of the table was reached.

list—An array of up to 400 acp_summary_rec structures (prototyped as acp_list_rec) receives the data requested. It must be dimensioned large enough to for the number of records requested (1 + end_rec - start_rec).

Note that the acp_summary_rec structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

`cm50$m_handler`
`cm50$m_msgon`

20.2 PROGRAMMATIC INTERFACE TO DDT OPERATIONS

All CM50S DDT operations except for `Edit` can be accessed through the Programmatic Interface. All Programmatic Interface routines are called as functions, and the status of each call is returned as the value of the function call. The calling program must include the CM50_FLAGS_INCLUDE file and the cm50_ddt_INCLUDE file. Both files must be appropriate to the language being used (see heading 2.9.1).

Exception handling is provided by standard VMS condition handling routines or by custom routines written by the user. The Programmatic calls for all DDT functions are described in detail in the following paragraphs.

20.2.1 Build/Rebuild DDT

This routine is called to build, or rebuild, a DDT binary file from a DDT source file. Flag options include CG residence for the DDT and DDT Rebuild.

20.2.1.1 Example "C" Call for Build/Rebuild DDT

```
return_status = cm50_ddt_build
                (ddt_name,
                 source_path,
                 &cg_port_number,
                 description,
                 &flags);
```

20.2.1.2 Parameter Definitions for Build/Rebuild DDT

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—The name (9-character string, prototyped as ddt_name_type) of the DDT to be used to retrieve data. It may be left blank if the full source path is specified.

source_path—Full pathname (80-character string, prototyped as path_name_type) of the DDT source file. If set to spaces, the default is the DDT name in the current directory.

cg_port_number—A short integer which specifies which CG (1-4) the DDT is associated with.

description—A text description (fixed-length 36-character array prototyped as ddt_desc_type) of the DDT being built. Note that if the DDT source file specifies a description, that description will be used and the value of this argument is ignored.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
cm50$m_rebuild_ddt
cm50$m_dmp_ddt_errors
cm50$m_no_source_debug
cm50$m_cg_res
cm50$m_write_vt
```

NOTE

If the DDT (or another DDT by the same name) has already been built, then the cm50$m_rebuild_ddt flag must be set ON.

For a new DDT, the cm50$m_rebuild_ddt flag must be OFF.

20.2.2 Delete DDT

This routine is called to delete a DDT that already exists in the DDT table. If the DDT is installed in the CG, it is removed.

20.2.2.1 Example "C" Call for Delete DDT

```
return_status = cm50_ddt_delete
                (ddt_name,
                 &flags);
```

20.2.2.2 Parameter Definitions for Delete DDT

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—(9-character string, prototyped as ddt_name_type) of the DDT to be deleted.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

20.2.3 Get DDT Summary

This routine is called to summarize the specifications of a particular DDT.

20.2.3.1 Example "C" Call for Get DDT Summary

```
return_status = cm50_ddt_sum
                (ddt_name,
                 &summary,
                 &flags);
```

20.2.3.2 Parameter Definitions for Get DDT Summary

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—(9-character string, prototyped as ddt_name_type) of the DDT that is to be summarized.

summary—This argument (declared as type ddt_summary_rec) receives the requested information. Its contents are:

- Name of the DDT being summarized
- Pathname of the DDT's source file
- Description of the DDT
- Date that the DDT was first built
- Name of the original builder
- Most recent time the DDT was modified
- Installation status of the DDT
- Number of points in the DDT
- DDT Type—Input, Generic Input, Output, Generic Output, or History
- CG number that the DDT is associated with
- Whether or not DDT is installed in CG
- Name of ACIDP DDT is connected to
- Prefetch triggers

Note that the ddt_summary_rec structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

`cm50$m_handler`
`cm50$m_msgon`

20.2.4 Get List of DDTs

This routine is called to retrieve a list of DDT summaries. Up to 500 DDT summaries may be returned in a single call. If more DDTs are installed on your system, they may all be retrieved by repeating this all with start_record set to 1, 501, 1001 and 1501 on successive calls.

20.2.4.1 Example "C" Call for Get List of DDTs

```
return_status = cm50_ddt_list
                (&start_record,
                 &end_record,
                 &count,
                 &list,
                 &flags);
```

20.2.4.2 Parameter Definitions for Get List of DDTs

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

start_record—This short integer specifies the number of the first DDT to retrieve.

end_record—This short integer specifies the number of the last DDT to retrieve.

count—This short integer receives the actual number of DDT records returned to the caller.

list—This argument will receive an array of DDTs information. Declare as an array of up to 500 ddt_summary_rec (prototyped as ddt_summary_arr). The information returned for each record is

- Name of the DDT being summarized
- Pathname of the DDT's source file
- Description of the DDT
- Date that the DDT was first built
- Name of the original builder
- Most recent time the DDT was modified
- Installation status of the DDT
- Number of points in the DDT
- DDT Type—Input, Generic Input, Output, Generic Output, or History
- CG number that the DDT is associated with
- Tells whether DDT is installed in CG
- Name of ACIDP DDT is connected to

Note that the ddt_summary_rec structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

`cm50$m_handler`
`cm50$m_msgon`

20.2.5 Get DDT Detail

This routine is called to retrieve the detail information for the named DDT.

20.2.5.1 Example "C" Call for Get DDT Detail

```
return_status = cm50_ddt_detail
             (ddt_name,
              &summary,
              &data,
              &points,
              &details,
              &values,
              &flags);
```

20.2.5.2 Parameter Definitions for Get DDT Detail

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—(9-character string, prototyped as ddt_name_type) of the DDT being summarized.

summary—A record (declared as type: ddt_summary_rec) containing:

`DDT_Name`	Name of the DDT being summarized
`DDT_Sourc_Loc`	Pathname to the DDT's source file
`DDT_Desc`	Description of the DDT
`Built_On`	Date that the DDT was first built
`Built_By`	Tells who the original builder was
`Modified_On`	Most recent time the DDT was modified
`DDT_Status`	Installation status of the DDT
`Number_of_Pts`	Number of points in the DDT
`DDT_Type`	Input, Output, or History DDT
`CG_Port_Num`	CG that the DDT is associated with
`In_CG`	Tells whether DDT is installed in CG

Note that the ddt_summary_rec structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

data—A record (declared as type ddt_data_type) containing:

`DDT_Types`	Names data types found in the DDT
`TTL_Each_Type`	Counts for each data type found

points—An array of up to 300 points_rec records (one record for each point in the DDT -- prototyped as points_arr) containing:

`Point_Name`	Point name
`Param_Name`	Parameter name (with index)

Note that the points_rec structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

details—An array of up to 300 detail_rec records (one per point -- prototyped as detail_rec_arr) containing:

`Process_Type`	Real, Integer, ASCII, Enumeration, or Ordinal
`Dest_Src`	Destination or Source offset value
`Test`	Use test Y/N and test data value
`BVS`	Bad value substitution Y/N and data
`Algo`	Algorithm number selection and data
`Limits`	Limit check Y/N and data

values—An array of up to 300 subst_type records (one per point -- prototyped as values_arr). This argument will contain useful information only if full Table Processing (including a Values Table) is being used with the DDT. It contains the values from the last use of the DDT showing the values before and after table processing conversions. Any LCN Real data Bad Values are returned as zeros.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

20.2.6 Connect DDT to ACIDP

This routine is called to connect a DDT to an ACIDP for the purpose of enabling the Data Prefetch Function in the CG. The ACIDP-ACP connection must already exist and the DDT must be CG-resident and not already connected to an ACIDP.

The `ddt_name`, and either the `acp_name`, or `acidp_name` parameters are required in the call. The Schedule, PPS and Demand parameters also are required.

20.2.6.1 Example "C" Call for Connect DDT to ACIDP

```
return_status = cm50_ddt_connect
                (ddt_name,
                 acidp_name,
                 acp_name,
                 &trigger,
                 &flags);
```

20.2.6.2 Parameter Definitions for Connect DDT to ACIDP

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—(9-character string, prototyped as ddt_name_type) of the DDT that is to be connected to an ACIDP.

acidp_name—(16-character string, prototyped as cm50_long_acidp) of the ACIDP to which the DDT is to be connected. The `acidp_name` can be blanks if a valid `acp_name` is provided.

acp_name—(12-character string, prototyped as acp_name_type) of the ACP connected to the ACIDP to which the DDT is to be connected. The `acp_name` can be blanks if a valid `acidp_name` is provided

trigger—single character code with the three high-order bits assigned these meanings:

- Bit 7 : `Schedule`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 6 : `PPS` (Point_Process_Special)—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 5 : `Demand`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

20.2.7 Disconnect DDT from ACIDP

This routine is called to disconnect a DDT from an ACIDP. At least one of the three parameters, `ddt_name`, `acp_name`, or `acidp_name` is required in the call (the others are passed as blanks). The ACIDP-ACP-DDT connection must already exist.

20.2.7.1 Example "C" Call for Disconnect DDT from ACIDP

```
return_status = cm50_ddt_disconnect
                  (ddt_name,
                   acidp_name,
                   acp_name,
                   &flags);
```

20.2.7.2 Parameter Definitions for Disconnect DDT from ACIDP

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—(9-character string, prototyped as ddt_name_type) of the DDT that is to be disconnected. Can be blanks if either `acidp_name` or `acp_name` contains a valid name.

acidp_name—(16-character string, prototyped as cm50_long_acidp) of the ACIDP from which the DDT is to be disconnected. Can be blanks if either `ddt_name` or `acp_name` contains a valid name.

acp_name—(12-character string, prototyped as acp_name_type) of the ACP connected to the ACIDP from which the DDT is to be disconnected. Can be blanks if either `ddt_name` or `acidp_name` contains a valid name.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

20.2.8 Modify Triggers

This routine is called to modify the Triggers associated with a DDT that is connected to an ACIDP. At least one of the three parameters, `ddt_name`, `acp_name`, or `acidp_name`, is required in the call (the others are passed as blanks). The ACIDP-ACP-DDT connection must already exist.

20.2.8.1 Example "C" Call for Modify Triggers

```
return_status = cm50_ddt_triggers
                (ddt_name,
                 acidp_name,
                 acp_name,
                 &trigger,
                 &flags);
```

20.2.8.2 Parameter Definitions for Modify Triggers

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—(9-character string, prototyped as ddt_name_type) of the DDT that is connected to the specified ACIDP. Can be blanks if either `acidp_name` or `acp_name` contains a valid name.

acidp_name—(16-character string, prototyped as cm50_long_acidp) of the ACIDP to which the specified DDT is connected. Can be blanks if either `ddt_name` or `acp_name` contains a valid name.

acp_name—(12-character string, prototyped as acp_name_type) of the ACP connected to the specified ACIDP. Can be blanks if either `ddt_name` or `acidp_name` contains a valid name.

trigger—A single character code with the three high-order bits assigned these meanings:

- Bit 7 : `Schedule`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 6 : `PPS` (Point_Process_Special)—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 5 : `Demand`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

20.2.9 Install DDT Into CG

This routine is called to install the DDT into the CG.

20.2.9.1 Example "C" Call for Install DDT Into CG

```
return_status = cm50_ddt_install
                 (ddt_name,
                  &flags);
```

20.2.9.2 Parameter Definitions for Install DDT Into CG

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—(9-character string, prototyped as ddt_name_type) of the DDT to be installed into the CG.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

`cm50$m_handler`
`cm50$m_msgon`

20.2.10 Uninstall DDT from CG

This routine is called to remove a DDT from the CG.

20.2.10.1 Example "C" Call for Uninstall DDT from CG

```
return_status = cm50_ddt_uninst
                 (ddt_name,
                  &flags);
```

20.2.10.2 Parameter Definitions for Uninstall DDT from CG

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—(9-character string, prototyped as ddt_name_type) of the DDT to be removed from the CG.

flags—This short integer (prototyped as cm50_flag_type) sets options as described in section 17.1.3. The following flags apply to this call:

`cm50$m_handler`
`cm50$m_msgon`

20.3 PROGRAMMATIC INTERFACE TO CG DATABASE

These functions return information about the current points configured in the database of any CG connected to the CM50. The language specific declarations for these functions are contained in `CM50_CGDATA_INCLUDE.H`

20.3.1 Resident DDT Summary

This function returns a list of all of the DDTs currently resident in the CG.

20.3.1.1 Example "C" Call for Resident DDT List

```
return_status =  cm50_cg_rddt
                 (&cg_port_num,
                  &number_of_values,
                  ddt_list);
```

20.3.1.2 Parameter Definitions for Resident DDT List

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of an short integer identifying the CG to be accessed.

number_of_values—The name of a short integer that returns the number of DDTs currently installed as resident in the CG.

ddt_list—The name of an array of up to 40 cm50_resddt character-arrays (prototyped as cm50_resddt_list) that will contain the names of the resident DDTs.

20.3.2 Calculated Results Data Points List

This function returns a list of all of the CRDPs currently configured in the CG.

20.3.2.1 Example "C" Call for CRDP List

```
return_status =  cm50_cg_crdp
                 (&cg_port_num,
                  &number_of_values,
                  crdp_list);
```

20.3.2.2 Parameter Definitions for CRDP List

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of a short integer identifying the CG to be accessed.

number_of_values—The name of a short integer that returns the number of CRDPs currently configured in the CG.

crdp_list—The name of an array of up to 500 cm50_resacidp character-arrays (prototyped as cm50_crdp_list) that will contain the names of the CRDPs.

Note that the cm50_resacidp structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

20.3.3 ACIDP Detail

This function returns a information about the current status of a specific ACIDP.

20.3.3.1 Example "C" Call for ACIDP Detail

```
return_status =  cm50_cg_adetail
                 (&cg_port_num,
                  &acidp_record);
```

20.3.3.2 Parameter Definitions for ACIDP Detail

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of a short integer identifying the CG to be accessed.

number_of_values—The name of a short integer that returns the number of DDTs currently installed as resident in the CG.

acidp_record—The name of a record (declared as type cm50_acidp_rec) with the following format:

ACIDP	: 8-character name of the ACIDP
DESC	: 24-character descriptor of the ACIDP
UNIT	: 2-character LCN Unit to which the ACIDP is assigned
KEYWORD	: 8-character LCN alias for the ACIDP
ACP	: 12-character name of the connected ACP
MODE	: 8-character enumerated value of the Program Mode
EXEC	: 8-character enumerated value of the Execution State
ACCES	: 8-character enumerated value of the Data Access Mode
DDT	: 9-character name of attached DDT
ACTYP	: 8-character enumerated value of the Activation Type
INHIB	: 8-character enumerated value of the Inhibit flag
STIME	: 8-character value of the Scheduled Start Time
PERIOD	: 8-character value of the Schedule Cycle Period
NXTTIM	: 18-character value of the Next Scheduled Activation Time
TAKEIP	: 4-character enumerated value of the Take_Initial_Path flag
RUNINIT	: 4-character enumerated value of the Run_on_Initialization flag
CONFWT	: 4-character enumerated value of the Confirm_Wait flag
CONFRQ	: 4-character enumerated value of the Confirm_Request flag
SCH	: 4-character enumerated value of the Schedule Activation flag
PPS	: 4-character enumerated value of Program_Special Activation flag
DMD	: 4-character enumerated value of Operator_Demand Activation flag
GROUP	: unsigned short integer value of the Group code.

Note that the cm50_acidp_rec structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

20.3.4 ACIDP Summary

This function returns a list of all of the ACIDPs configured in the CG.

20.3.4.1 Example "C" Call for ACIDP Summary

```
return_status =  cm50_cg_acidp
                 (&cg_port_num,
                  &number_of_values,
                  acidp_list,
                  acp_list,
                  mode_list,
                  state_list,
                  ddt_list,
                  trigger_list);
```

20.3.4.2 Parameter Definitions for ACIDP Summary

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_port_num—The name of a short integer identifying the CG to be accessed.

number_of_values—The name of a short integer that returns the number of ACIDPs currently configured in the CG.

acidp_list—The name of an array of up to 250 cm50_resacidp character-arrays (prototyped as cm50_acidp_list) that will contain the names of the resident ACIDPs.

Note that the cm50_resacidp structure can vary for different releases of CM50S, so programs using this call should be recompiled when CM50S is upgraded.

acp_list—The name of an array of up to 250 cm50_resacp character-arrays (prototyped as cm50_cgacp_list) that will contain the names of the ACPs connected to the corresponding ACIDP.

mode_list—The name of an array of up to 250 short integers (prototyped as cm50_cgenum_list) that will contain the integer code for the installation mode of each ACIDP.

state_list—The name of an array of up to 250 short integers (prototyped as cm50_cgenum_list) that will contain the integer code for the current execution state of each ACIDP.

ddt_list—The name of an array of up to 250 cm50_resddt character-arrays (prototyped as cm50_cgddt_list) that will contain the names of the DDT (if any) connected to the corresponding ACIDP.

trigger_list—The name of an array of up to 250 cm50_restrig records (prototyped as cm50_trig_list),where each record is an array of 3 Boolean short integer values (indicating by 1 or 0 whether or not the connected DDT will be prefetched when the ACIDP is triggered on Schedule, Operator_Demand, or PPS).

20.3.5 LCN Configuration

These functions return information about the LCN configuration parameters for a specified CG. Note that the specified CG must be running TDC 3000 release 400 or later for this function to get a successful return_status.

20.3.5.1 Example "C" Call for LCN Configuration

```
return_status = CM50_CG_CONFIG
                (&cg_prot_num,
                 &cgconfig_record);
```

20.3.5.2 Parameter Definitions for LCN Configuration

return_status—The name of an Integer to receive the overall return status of the function call. For fully successful calls, return_status = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

cg_prot_number—The name of a short integer identifying the CG (1 to 4) to be accessed.

cgconfig_record—The name of a record (declared as type cm50_cgconfig_rec) with the following fields:

LCN_VER	: short integer TDC 3000 software release level
LCN_REV	: short integer TDC 3000 revision level (future)
LCN_NODE	: short integer LCN node number of this CG
CG_VER	: short integer CG personality software release
CG_REV	: short integer CG personality software revision
TIME_SYNCH	: short integer CG Time synchronization period
CONFIRM_TIME	: short integer CG Time out for message confirm
CG_STATION	: short integer HDLC station number of the LCN
T1_TIME	: short integer T1 timer
N2_COUNT	: short integer Retry count
FLOAT_FORMAT	: short integer Floating point format (should be 2 for IEEE)
BAUD_RATE	: short integer enumeration (0= 1200, 1= 1760, 3= 2400, 5= 4800, 7= 9600, 8= 19200, 13= 38400, 14= 56700, 15= 76800)
TAG_SIZE	: short integer (0 for 8-character maximum, 1 for 16-characters)
HM_USER_MIN	: short integer number of minutes in a user average
HM_SHIFT_WK	: short integer number of shifts per week
HM_START_HR	: short integer daily/weekly averages starting hour (0 starts Sunday morning just after midnight)
HM_MONTH_TYP	: short integer (0 is calendar, 1 is 28-day cycles)
PINID	: 2-character identifier of this LCN for Network Gateway routing
DESCR	: 40-character CG descriptor on the LCN

20.4 PROGRAMMATIC INTERFACE TO FILE TRANSFER

These functions execute LCN file transfer commands programmatically. The calling program must include the CM50_FLAGS_INCLUDE.H and CM50_FTF_INCLUDE.H files in its source to insure that the functions and arguments are properly declared.

The dataout facility allows the user, when requesting the execution of specific file transfer transactions, to place relevant data in the dataout or catalog file. This dataout file is a shared file by all concurrent users of file transfer. For example, user "Jones" requests a CM50_FILE_CATALOG transaction, the results of which are placed into the current dataout file. User "Smith" then requests a CM50_VOLUME_CATALOG transaction. These results also are placed into the same (current) dataout file.

CM50_FILE_CATALOG and CM50_VOLUME_CATALOG are the only file transfer transactions that require a dataout file. Other file transfer transactions treat dataout as an option for journalizing activity.

20.4.1 Read LCN File

This procedure will transfer a single file from an LCN NET volume to CM50S. Wildcard transfers of files are not supported. This procedure will also create an "LCN ATTRIBUTES" file for every LCN file that is transferred. Multiple copies of the same file, within the same VMS directory, are not allowed. The version number of the attributes file should remain 1. For more information regarding file attributes refer to the WRITE file procedure.

20.4.1.1 Example "C" Call for File Read

```
return_status  =  cm50_lcn_read
                     (lcn_file,
                      host_file,
                      acidp_name,
                      &cg_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.1.2 Parameter Definitions for File Read

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx.

host_file—VMS pathname (packed array of 80 characters, prototyped as PATH_NAME_TYPE) to be used to store the LCN file (and its asscociated attributes file). If no extension is specified, the VMS default of .DAT will be used. If no directory is specified, the user's current default directory will be used. The LCN attributes file will use the following naming convention: the filename suffix or extension will be preceded by an under-bar character, and followed by a period "LA" extension. For example; the LCN filename of FORMULAE.CL would have an attribute file of FORMULAE_CL.LA. Note: The transfer will fail if the pathname matches that of an existing file.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.2 Write LCN File

This procedure will transfer a single file from CM50S to LCN NET volume. This procedure requires the LCN ATTRIBUTES file for every LCN file that is transferred. Multiple copies of an LCN FILE within the same VMS directory are allowed. These files would have been created by modifying the original LCN FILE which was transferred as version 1. The version number of the attributes file should be 1.

20.4.2.1 Example "C" Call for File Write

```
return_status  =  cm50_lcn_write
                     (host_file,
                      lcn_file,
                      acidp_name,
                      &file_code,
                      &cg_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.2.2 Parameter Definitions for File Write

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

host_file—VMS pathname (packed array of 80 characters, prototyped as PATH_NAME_TYPE) of the file to be transferred to the LCN If no directory is specified, the user's current default directory will be used. The associated LCN attributes file (with an extension of .LA) must be in the same directory.

lcn_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

file_code—Name of a shortword (prototyped as write_file_type) that determines whether the LCN file is to be replaced if it already exists at the LCN NET volume. The default is to abort the write if the file already exists. The enumerated values are:

`replace_write` = 0: Replace existing file
`abort_write` = 1: Return an error if the file already exists.

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.3 List LCN File Attributes

This request will retrieve the file attributes for a specific LCN file. Wildcard characters and dataout options are not permitted. The file attributes are :

- Lcn file type—contiguous or linked
- Lcn file protection
- Record size
- Block size
- Lcn file configuration
- Lcn file revision
- Directory timestamp (Mo/Dd/Yr Mm:Ss)
- Logical number of blocks
- Logical number of records
- File Descriptor
- Starting Sector
- Ending Sector

20.4.3.1 Example "C" Call for File Attributes

```
return_status  =  cm50_attr_list
                     (lcn_file,
                      acidp_name,
                      &attributes,
                      &cg_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.3.2 Parameter Definitions for File Attributes

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

attributes—Buffer (prototyped as file_attribute_block, and described in CM50$LIB:CM50_FTF_INCLUDE.H) that will receive requested data.

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.4 List LCN File Names

This transaction will retrieve up to 1180 file names and extensions from an LCN NET volume. If the number of files exceeds the buffer capacity of 1180, then multiple requests by directory, file type extension, or filename syntax must be used. Wildcards are permitted.

20.4.4.1 Example "C" Call for List Files

```
return_status  =  cm50_file_list
                     (lcn_file,
                      acidp_name,
                      &file_list,
                      &cg_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.4.2 Parameter Definitions for List Files

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:
NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

file_list—Buffer (prototyped as file_list_array, and described in CM50$LIB:CM50_FTF_INCLUDE.h) that will receive the list of file names and attributes.

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.5 List LCN Volumes/Directories

This transaction will fetch from the History Module volume and directory names and sector usage figures. Wildcards and options are not permitted for this transaction type.

20.4.5.1 Example "C" Call for List Volumes

```
return_status  =  cm50_hm_listf
                     (lcn_device,
                      acidp_name,
                      &vol_record,
                      &cg_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.5.2 Parameter Definitions for List Volumes

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_device—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the device to be cataloged. Use the form PN:**nn** where **nn** is the LCN node number.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

vol_record—Buffer (prototyped as volume_structures and described in CM50$LIB:CM50_FTF_INCLUDE.h) that will receive the Volume and directory information. This information includes:

- Number of Volumes
- Number of Sectors / Device
- Sectors in Use / Device
- Volume Name(s)
- Directory Name(s) on each volume

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.6 Cataloging LCN Fles to Dataout

This file transfer transaction will list the LCN FILE ATTRIBUTES of one or more files into the current dataout file. The dataout file must have been previously established. The absence of a dataout specification will result in an error return. Further processing requires that the dataout or catalog file be transferred to the VAX using the CM50_LCN_READ programmatic function.

20.4.6.1 Example "C" Call for File Catalog

```
return_status =  cm50_file_catalog
                      (lcn_file,
                       cat_file,
                       acidp_name,
                       &cg_port_number,
                       &lcn_sts,
                       &flags);
```

20.4.6.2 Parameter Definitions for File Catalog

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:

NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

Optional suffixes will increase the amount of information returned:

-FD will cause file descriptors to be listed
-REC will cause record and block data to be listed

cat_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to receive the catalog. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of an unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.7 Cataloging LCN Volumes to Dataout

This file transfer transaction will list the LCN Volumes and Directories for all History Modules on the NET or for a specific History Module. The dataout file must have been previously established. The absence of a dataout specification will result in an error return.

20.4.7.1 Example "C" Call for Volume Catalog

```
return_status  =  cm50_volume_catalog
                     (lcn_device,
                      cat_file,
                      acidp_name,
                      &cg_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.7.2 Parameter Definitions for Volume Catalog

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_device—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the device to be cataloged. Use the form NET or PN:**nn** where **nn** is the LCN node number.

cat_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to receive the catalog. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.8 LCN File Copy

This file transfer transaction will copy a single file or all files from one NET volume to another Net volume. The -D option is supported for journalizing all copies to a dataout file. The dataout file must have been previously established. Wildcards are permitted; however, the destination suffix must always be the same as the source suffix. Note that using the -D option without having previously defined a dataout path will result in an error and the copy function will not have been completed.

20.4.8.1 Example "C" Call for LCN File Copy

```
return_status  =  cm50_lcn_copy
                     (lcn_file,
                      out_file,
                      acidp_name,
                      &cg_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.8.2 Parameter Definitions for LCN File Copy

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:
NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

out_file—Recipient filename (28-character string prototyped as type lcn_path_name) specifying the pathname of the new file. The actions will be journalized if a DATAOUT file has been enabled and the " -D" option suffix is appended to the filename.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.9 LCN File Move

This file transfer transaction will move a single file or all files from one directory to another directory within the same NET volume. Wildcards are permitted and the -D option is supported for journalizing all moves to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the move function will not have been completed.

20.4.9.1 Example "C" Call for LCN File Move

```
return_status  =  cm50_lcn_move
                     (lcn_file,
                      out_directory,
                      acidp_name,
                      &cg_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.9.2 Parameter Definitions for LCN File Move

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:
NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

out_directory—Directory name (28-character string prototyped as type lcn_path_name) specifying the directory to receive the moved file. This directory must be on the same HM volume as the original file. (The file name and extensions will remain unchanged.) The actions will be journalized if a DATAOUT file has been enabled and the" -D" option suffix is appended to the filename.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.10 LCN File Rename

This file transfer transaction will rename a single file or all files on the History Module. Wildcards are permitted and the -D option is supported for journalizing all renames to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the rename function will not have been completed.

20.4.10.1 Example "C" Call for LCN File Rename

```
return_status  =  cm50_lcn_rename
                     (lcn_file,
                      out_file,
                      acidp_name,
                      &cg_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.10.2 Parameter Definitions for LCN File Rename

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:
NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

out_file—Recipient filename (28-character string prototyped as type lcn_path_name) specifying the pathname of the new file. (The directory and extensions will remain unchanged.) The actions will be journalized if a DATAOUT file has been enabled and the" -D" option suffix is appended to the filename.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.11 LCN File Delete

This file transfer transaction will delete a single file or all files from the specified volume on the History Module. Wildcards are permitted and the -D option is supported for journalizing all deleted files to a dataout file. The dataout file must have been previously established. Note that using the -D option without having previously defined a dataout path will result in an error and the delete file function will not have been completed. Once deleted the file is cannot to be recovered.

20.4.11.1 Example "C" Call for LCN File Delete

```
return_status  =  cm50_lcn_delete
                     (lcn_file,
                      acidp_name,
                      &g_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.11.2 Parameter Definitions for LCN File Delete

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to be transferred from the LCN. Use the form NET>VDIR>FILENAME.xx. Wildcards (*) are permitted for the file name and/or extension. Formats:

NET>VDIR>*.*
NET>VDIR>FILENAME.*
NET>VDIR>*.nn

The actions will be journalized if a DATAOUT file has been enabled and the "-D" option suffix is appended to the pathname.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.12 LCN Directory Maintenance

These file transfer transactions will create or delete a directory under a volume on the History Module. No wildcard characters or options are permitted.

20.4.12.1 Example "C" Call for Directory Maintenance

```
return_status = cm50_lcn_directory
                        (lcn_directory,
                         &action,
                         acidp_name,
                         &cg_port_number,
                         &lcn_sts,
                         &flags);
```

20.4.12.2 Parameter Definitions for Directory Maintenance

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

lcn_directory—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the LCN directory to be created or deleted. Use the form NET>VDIR> DIR (Note the space delimiter before the directory name.)

action—A shortword (prototyped as dir_func_type) that specifies whether the named directory is to be created or deleted. The enumerated values are:

```
create_directory = 0
delete_directory = 1
```

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.13 Dataout Status

The dataout function allows the user, when requesting the execution of specific file transfer transactions, to place relative data in the dataout or catalog file. This dataout file is a shared file by all concurrent users of file transfer. For example, user "Jones" requests a CM50_FILE_CATALOG transaction, the results of which are placed into the current dataout file. User "Smith" then requests a CM50_VOLUME_CATALOG transaction. These results also are placed into the same (current) dataout file. The dataout file may be transferred to the VAX host using a CM50_LCN_READ request. The CM50_DATA_OUT transaction is provided to enable, disable, or query the file transfer dataout status.

20.4.13.1 Example "C" Call for DATAOUT status

```
return_status  =  cm50_data_out
                     (cat_file,
                      acidp_name,
                      &do_action,
                      &cg_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.13.2 Parameter Definitions for DATAOUT status

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

cat_file—LCN pathname (28-character string prototyped as type lcn_path_name) identifying the file to receive the catalog. Use the form NET>VDIR>FILENAME.xx.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

do_action—A shortword (prototyped as do_func_type) that specifies the action to be taken. The values are:

`disable` = 0: Disable dataout journaling
`enable` = 1: = Enable dataout journaling using the specified path
`request_status` =2: Return the current dataout path

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

20.4.14 Abort File Transfer Transaction

The transaction CM50_ABORT_TRANSFER will terminate the current transaction in progress. The initiator of the transaction will receive a CM50_FTF_ABORT error return status. The initiator of the CM50_ABORT_TRANSFER request will receive a normal return status. No error is generated if there is not a current process to abort.

20.4.14.1 Example "C" Call for Abort Transfer

```
return_status  =  cm50_abort_transfer
                     (acidp_name,
                      &cg_port_number,
                      &lcn_sts,
                      &flags);
```

20.4.14.2 Parameter Definitions for Abort Transfer

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status = 1`. See Appendix A.2 for an explanation and listing of all assigned return code values.

acidp_name—A 16-character string (prototyped as acidpoint_type) reserved for future security use. This field should be set to all spaces.

cg_port_number—The name of a unsigned short integer (prototyped as cg_num_type) that specifies which Computer Gateway (1-4) will be used for access to the LCN.

lcn_sts—The name of a short integer which will receive the detailed error code from the LCN if the overall return_status is cm50_ftf_filmgr (215004012) or cm50_ftf_utility (215004146). See Appendix A.4 for specific meanings.

flags—Integer parameter (optionally prototyped as cm50_flag_type) that sets options as described in section 17.1.3. The cm50$m_handler option is the only flag applicable to File Transfer functions.

STATUS AND ERROR CODES
Appendix A

This appendix lists the status codes that are returned following data exchanges.

A.1 DATA ACCESS STATUS CODES

The following LCN data access status codes apply to the individual point.parameter values and are included in the data returned to the ACP by the user interface routines (variously as STATUS_TABLE, VALUE_STATUS, STORE_STATUS or DATA_ACCESS_STATUS values).

Code	Explanation
1	Value out of range, clamped value stored and clamped value returned
2	String was too long, truncated value was stored
3	Initialization warning
4	End of history-data file
5	End of user-specified samples or time
6	User-allocated buffer is full
7	Spare warning 05
8	Data item is valid
9	Access level invalid (HG)
10	Invalid algorithm for controller type
11	Algorithm must be DDC (tried op store)
12	Algorithm must be SPC (tried sp store)
13	Value type presented for store was not the expected type
14	Bias is undergoing initialization
15	Both init and tracking cannot be configured
16	HG box status is failed
17	HG box not configured
18	Mode must be manual
19	Cannot convert numeric variable to IEEE
20	Cannot convert numeric variable to JFP
21	Cannot reset timer unless stopped
22	Cascade request flag must be set
23	Change not permitted by operator
24	Cascade enable flag must be set (for change to computer mode)
25	Configuration mismatch error
26	Control output connection not configured
27	Current mode disallows mode change
28	Current mode disallows store
29	Failure of device where entity resides
30	Device where entity resides is in reset
31	Engineering unit span is too small
32	Given subscript is not implemented in entity
33	External mode switching is enabled
34	External switching option is not selected
35	Fatal error: no match for case selector
36	HG hiway status is failed

37 Hiway access failure
38 Illegal value
39 Init cannot be configured
40 Init must be configured
41 Value cannot be changed because of initialization
42 Invalid internal variable number
43 Invalid algorithm id in base segment
44 Invalid algorithm equation
45 Invalid control algorithm id error
46 Invalid destination parameter
47 Invalid destination point id
48 Invalid mode
49 Invalid mode attribute
50 Invalid pb option
51 Invalid point for point build
52 Invalid sp option
53 Invalid target value processor state
54 Max structural parameter exceeded error
55 Mode attribute error
56 Invalid PV control algorithm combination
57 Duplicate batch id
58 Current mode does not allow parameter value to be changed
59 Mode illegal for this point type
60 Mode keylock error
61 Mode not allowed with configured algo
62 Mode not currently legal
63 Mode not man or point active
64 Must have digital output
65 Currently there is no value for this parameter
66 Nocopts must be zero to change coptype
67 Normal attribute is not configured
68 Normal mode is not configured
69 Value cannot be changed, because parameter is not selected
70 Number of outputs must exceed zero
71 Entry only possible in off-process personality
72 Output is undergoing initialization
73 Operator override is required for store
74 Parameter cannot be changed
75 Parameter index is invalid
76 Parameter is invalid
77 Parameter is undergoing initialization
78 Parameter is not configured
79 Parameter is not valid for the configured control algorithm
80 Parameter is not valid for the configured pv algorithm
81 Status of entity is partial error
82 Point must be inactive
83 Point is red tagged
84 Point is undergoing initialization
85 Point is not available to point build
86 Definition of point is not complete enough to make point valid
87 Point not secondary
88 Point status error (HG status is in error)
89 Point type is invalid (HG configuration error)
90 PPS only active points error

91 Process box restore in operation
92 Process module off
93 PV source disallows PV change (HG cannot change PV)
94 Ratio is undergoing initialization
95 Read only parameter (HG cannot write)
96 Red tag error (all output entries are blocked)
97 Secondary has too many primaries
98 Secondary must be off node
99 Secondary point has no primary
100 HG slot number invalid
101 Source of request is invalid (HG change is prohibited)
102 Source of Y is not configured (HG change to casc mode is invalid)
103 SP is undergoing initialization
104 SP is undergoing pv tracking
105 Store not allowed with spc connection
106 Store not permitted
107 Store not permitted from off point
108 Subscript error
109 System error (HG data handler detected)
110 Tracking cannot be configured
111 Entry prohibited because of tracking
112 Tracking must be configured
113 Spare error 3
114 Y cannot be configured to own LSP
115 Point type id error
116 Segment class error
117 Invalid control algorithm error
118 Invalid pv algorithm error
119 Segment size exceeds maximum segment size
120 Memory unavailable
121 Get memory error
122 Structural parameter missing
123 Structural parameter maximum violated
124 Invalid cc rank
125 Store not permitted while casreq
126 Analog output used for modulating control
127 Y must be configured to own lsp
128 Store not permitted unless free variable
129 Control lock error
130 Point not primary
131 Off node parameter access error
132 Memory allocation error
133 Unit mismatch
134 Point must be scheduled
135 Before after period incompatible
136 Integer value required
137 Invalid PV equation
138 Invalid ctl equation
139 Invalid number of PV inputs
140 Invalid number of ctl inputs
141 Invalid number of ctl outputs
142 Invalid pv algorithm id
143 Invalid ctl algorithm id
144 Invalid pv algorithm

145 Invalid ctl algorithm
146 Invalid rb option
147 PV algorithm id cannot be null
148 CTL algorithm id cannot be null
149 PV algorithm id structural parameter fetch error
150 CTL algorithm id structural parameter fetch error
151 RB option structural parameter fetch error
152 SP option structural parameter fetch error
153 PV equation structural parameter fetch error
154 Control equation structural parameter fetch error
155 Number of pv inputs structural parameter fetch error
156 Number of ctl inputs structural parameter fetch error
157 Number of ctl outputs structural parameter fetch error
158 Copctype structural parameter fetch error
159 Structural parameter fetch error
160 Invalid structural parameter
161 Configuration error
162 Number of ctl outputs is zero
163 Fetch not permitted
164 Change restricted to engineering personality
165 Change restricted to on process personality
166 Access level error
167 Attribute id error
168 Function level error
169 Parameter id error
170 Parameter qualifier type error
171 Parameter type invalid
172 Prefetch item error
173 Data type error
174 Rev 20 disallows both SOPL and trend memory
175 Limit or range exceeded
176 Limit or range crossover
177 Inconsistent lrc for point build
178 Point build HG only
179 Must configure enough cl slots for standard actions
180 Point must be inactive or ready
181 Point not active or not configured in a History Group
182 No batch slot or no slot of sufficient size is available
183 Number of concurrent batches not established
184 Only active or inactive allowed
185 Prototype must have cls to be cloned
186 Only one dual output store permitted per request
187 Illegal timer state
188 Illegal counter state
189 Invalid number of ainpts
190 Invalid number of ordstns
191 Invalid number of orvals
192 Spare error 4
193 Alarm configuration mismatch
194 Control state read error
195 Control state basic error
196 Control state test error
197 Input and output required
198 Input and output box mismatch

199 Current state disallows operational state change
200 Configuration forcing in effect
201 Current procmod disallows store
202 Device must be in idle
203 Device must be in reset
204 Illegal box protocol
205 Invalid box status
206 Invalid characterization type
207 Invalid database index
208 Parameter not loaded
209 PV source invalid
210 Event overload
211 SP lock error
212 Improper access level
213 Invalid attribute id
214 No such entity id
215 No such function level in data owner
216 Initialization value
217 Communication path to data owner is disabled
218 Invalid parameter id
219 Improper parameter qualifier type
220 Given parameter type is invalid for the requested operation
221 Prefetch item number
222 Serial number in point id does not match
223 Value type presented for store was not of the expected type
224 Invalid store value supplied
225 Subscript out of range
226 Unable to perform specified limit check on store value
227 Value out of range, store not performed
228 Path to parameter cannot be completed
229 Spare error 14
230 Spare error 15
232 Invalid PV source
233 Event Overload
234 SP lock error
235 Improper access level
236 Invalid attribute ID
237 No such entity ID
238 Improper function level
239 Initialization value
240 Illegal No Good store—Enumeration member not found or out of range
241 Input status table error
242 Point not in history
243 Enum convert failure
244 SDE store not allowed
245 ACP value bad
246 Subst value bad
247 Bad Algo
248 Div by zero
249 Limit exceeded
250 Real value NaN
251 Data error
252 Wrong data type

253 Get set error
254 Get IDB item error
255 Directed no store (Not an error. Returned for all store data elements whose store code value is not 0 or 1.)
256 Not enough contiguous memory
257 Data point is too large
258 Bad data returned (input was bad)
259 Total value size exceeded (The sizes of data for the elements in this DDT add up to more than can be handled by LCN data access mechanisms. The DDT must be redesigned.)
260 Value envelope in user-supplied buffer is inconsistent with the value returned
261 Process Manager—PPX task not terminated
262 Process Manager—PMM scan rate must be null
263 Process Manager—PMM scan rate must not be null
264 Process Manager—WM invalid
265 Process Manager—WM insufficient data
266 Process Manager—WM invalid state
267 Process Manager—scan table overflow
268 Process Manager—calculator syntax error
269 Process Manager—calculator expression too big
270 Process Manager—permissive error
271 Process Manager—interlock error
272 Process Manager—UCN communication error
273 Process Manager—entity ID error
274 Process Manager—slot checkpoint exceeds available extension memory
275 Process Manager—invalid state pa status
276 Process Manager—invalid for secondary pa status
277 Process Manager—checksum bad
278 Process Manager—invalid sequence execution state
279 Process Manager—invalid process module state
280 Process Manager—abnormal handler not available
281 Process Manager—sequence program must be loaded
282 Process Manager—sequence program exceeds configured slot size
283 Process Manager—no confirmable message pending
284 Process Manager—fatal sequence code errors
285 Process Manager—communications error
286 Process Manager—UCN encode error
287 Process Manager—UCN reception error
288 Process Manager—UCN transmit error
289 Process Manager—communication error
290 UCN data unavailable, target node status error
16384 (CM50_NEGATIVE_OVERFLOW) Extremely low value has been clamped to -1.70e+38
16385 (CM50_POSITIVE_OVERFLOW) Extremely high value has been clamped to 1.70e+38
16386 (CM50_NEGATIVE_INFINITY) IEEE negative infinity value has been clamped to -1.70e+38
16387 (CM50_POSITIVE_INFINITY) IEEE positive infinity value has been clamped to 1.70e+38
16388 (CM50_NAN) Bad value has been returned as a legitimate custom data segment value
16389 (CM50_MINUS_ZER0) IEEE -0.0 converted to zero.

A.2 RETURN_STATUS CODES

Each User Interface or Programmatic Interface function returns a `return_status` doubleword integer value that reflects the overall return status of the function call. A "normal" successful completion is signaled by a `return_status` = 1 (SS$_NORMAL).

CM50S return_status codes are integrated with the VMS system error codes. All error codes generated specifically by CM50S have a nine-digit decimal representation that starts with the decimal digits "2150" (equivalent to 0CD0 in 8-digit hexadecimal). CM50S applications can also cause other error codes to be reported by VMS. Of particular interest are VMS protection/access violations that can prevent the CM50S code from being executed; these codes generally begin with the hexadecimal digits "1021." You should note that CM50S displays error codes in decimal notation while the operator log of VMS aborts uses hexadecimal notation.

The `return_status` value for a CM50S return that requires additional investigation always begins with the base value of 215000000. To this is added a 4-digit status return code (multiplied by 10) followed by a "severity" code. Thus, the returned status value of 215003202 is equivalent to a status return of 0320 with a severity code of 2.

Rule of Thumb: If the `return_status` value is not = 1 and is even, there is a problem; if odd, the call was successful but individual data items may need checking.

Another way of looking at the non-normal `return_status` value is: 2150 nnnn s, where nnnn = status return code

0001-0040 = LCN return codes
0051-0058 = LCN Send Message return (Rel 1 codes + 50
0061-0076 = Get_History returns (Rel 1 codes+ 60. Exception:
Get_History return = 8 is set to SS$_NORMAL)
0081-0089 = Point List Access-specific errors
0100-0135 = CM50S File Access errors
0200-0299 = Communication errors
0300-0399 = Format Conversion errors
0400-0699 = LCN File Transfer errors

and where s = severity code
0 = warning
1 or 3 = informational
2 or 6 = error or failure

The following list contains all assigned return code/severity code combinations along with the associated identifiers as found in the include-files:
CM50$LIB:CM50_ERROR_INCLUDE.FOR (or .PAS or .H).

The associated error messages may be retrieved using the DCL lexical function F$MESSAGE provided that the process has previously issued a SET MESSAGE CM50$LIB:CM50_ERROR_MSG command to link the CM50S message file with the system messages.

NOTE

User programs should reference specific error conditions by identifier names. These will remain constant even if future VMS releases change the underlying return_status numeric values.

return_ status Value	Identifier	Error Message Cause/Corrective Action
000000001	SS$_NORMAL	<none>
215000012	CM50_DDT_MISS	DDT incomplete or not found Named DDT or MPL file not found on disk or DDT was built with errors and is not useable.
215000026	CM50_DDT_HIST	DDT is not HISTORY type Named DDT was built for Input or Output, or MPL history call attempted for a list containing multiple data types.
215000036	CM50_LCN_FAIL	Unable to access LCN—datalink failure CG or communications link has failed or CM50_PKT_PROCESSOR has been aborted.
215000042	CM50_ACP_RUN	ACIDP not in RUN state or DDT not Input The ACIDP named in the call (or connected to the calling ACP if not named in the call) is not in RUN state because its ACP is not active or the ACIDP is concurrently processing another data access request, or DDT for a Get data request was built for Output or History.
215000051	CM50_LCN_PART	Returned Data includes errors One or more of the requested data elements had an error on the LCN, see the Status table entries to identify the specific problems.
215000066	CM50_LCN_POINT	ACIDP not found or not connected to ACP Named Point is not defined in the LCN database, or Named ACIDP is not in the CG that is being accessed, or Named ACIDP is not connected to an ACP.
215000076	CM50_ACP_STAT	ACP not installed or restricted Store failed because the ACP is not connected to an ACIDP, or ACP is installed in Restricted mode, blocking stores.
215000082	CM50_ACP_ACCE	ACCESS key is Read only or DDT is in use ACIPD is defined to be READ_ONLY, blocking stores, or DDT is concurrently in use by another ACP, or ACIDP is currently servicing another data request.
215000092	CM50_LCN_PRIOR	Invalid Priority - must be 1 or 2 Error in LCN Priority argument.
215000106	CM50_LCN_PARAM	Point or Parameter not found Requested Tag not found on the LCN.
215000116	CM50_LCN_PARTYP	Value Type does NOT match Parameter Type Named Parameter has a different data type than the requested value.
215000122	CM50_GET_MEM	Unable to Get Memory in VAX
215000132	CM50_LCN_GTX	Invalid CG Transaction code Attempt was made to use a new feature with an older release of TDC3000 that does not support it.
215000146	CM50_LCN_ARRAY	Array Size too small The size of the parameter array in the LCN is greater than the stated size of the array in the request. For a convert id request, this is an information code; it is a fatal error only when attempting to store an array.
215000156	CM50_LCN_WORD	ACP connected ACIDP not in Run State or Value not Word Aligned (vintage calls only)
215000162	CM50_ACP_CG	ACIDP not CG resident Named ACIDP does not exist within the CG being accessed

215000186 CM50_DDT_CONN DDT already connected to ACIDP
DDT already connected to an ACIDP.

215000196 CM50_DDT_DISC DDT is not connected to an ACIDP
DDT is not connected to the ACIDP.

215000202 CM50_ACP_DISC ACP is not connected to ACIDP
ACIDP -- ACP connection must exist before a DDT can be connected to it for Prefetch.

215000212 CM50_LCN_OPT Invalid Option for this CG Transaction

215000226 CM50_LCN_RANGE Data Type Out of Range
Error in Value_Type argument (must be between 1 and 20, prior to TDC release 300, maximum value is 10).

215000236 CM50_LCN_ZPAR Parameter Index must be zero
If Parameter is not an array, the index must be blank or zero.

215000252 CM50_LCN_ENUM Invalid Enumeration
No enumeration set is defined on the LCN for the named Parameter.

215000266 CM50_LCN_NOPAR Parameter name not defined on LCN
Named parameter is not defined anywhere on the LCN.

215000276 CM50_LCN_INDX Index Out of Range
Invalid value for Parameter Index argument

215000282 CM50_ACC_PRIOR Invalid data Access Priority

215000292 CM50_LCN_TYPE Invalid data Type code
Requested Value_Type is not accessible with this call.

215000306 CM50_LCN_PTIX Point Index not Zero
(Vintage calls only) Point Index argument must be set to 0.

215000316 CM50_LCN_MATCH Point does not have the named Parameter
Both Point and Parameter names are defined on the LCN but the named point does not have the requested parameter.

215000322 CM50_ACC_SIZE Array pointer/size too Large
The Parameter Index, array size, in the call is larger than the size of the array on the LCN.

215000332 CM50_DDT_TYPE DDT contains Invalid data Type

215000356 CM50_DDT_PURP DDT Purpose does Not match Request
DDT built for Input can be used only for Get Data calls; Generic DDTs can only be used with Generic calls, etc.

215000362 CM50_DDT_HDR DDT header does not match table content
Corrupt DDT file, or IDB argument of a MPL call is not pointing to a valid list.

215000372 CM50_DDT_DUP Data type repeated inside a DDT
Each Value_Type can appear only once in a DDT or MPL.

215000386 CM50_DDT_ENUM DDT contains both Enum & Ordinal types
A single DDT or MPL call cannot process both Enumerated and Ordinal values.

215000396 CM50_DDT_SIZE DDT contains too many points
A DDT or Point List may not transfer more than 300 current parameters, or 24 history parameters, on a single call.

215000402 CM50_LCN_HDR Header Size does not match Message Length
Corrupt DDT file, or IDB argument of a MPL call is not pointing to a valid list.

215000412 CM50_LCN_OVR Array size too large
Value of Parameter Index (or word 7 of the internal point id) is larger than the dimension of the array on the LCN.

215000426 CM50_LCN_CLOK Could not read internal clock
Internal error on the LCN in attempting to access the system clock.

215000436 CM50_LCNSTCOD Invalid Store Code
Value of store_cd argument is not a valid code for the requested data type.

215000442 CM50_LCN_DUP Duplicate Entity name
Error in LCN point name database.

215000506 CM50_MSG_NULL No message
No message is available for retrieval by Get Message call or the CG was unable to forward the message across the LCN on a Send Message call. (Check the Area Database on the LCN)

215000516 CM50_MSG_TIME Time out before confirmation received

215000521 CM50_MSG_TRUNC Received message was truncated
Calling arguments did not allocate enough space for the entire message or history values retrieved from the LCN.

215000532 CM50_MSG_CONF Invalid confirmation argument
Message confirmation must be 0 or 1.

215000546 CM50_MSG_WAIT Invalid wait time argument
Message time-out must be between 0 and 3600 seconds.

215000556 CM50_MSG_DEST Invalid destination
Message destination must be between 0 and 2.

215000561 CM50_MSG_QUE Message received & another one is queued
Successful call, another message is waiting.

215000571 CM50_MSG_QUET Message truncated & another one is queued
Requested message was longer than specified Message_size and another message is waiting.

215000586 CM50_MSG_SIZE Invalid message array size
Message Size must be between 1 and 120 characters.

215000612 CM50_HIS_MISS DDT not found
Named DDT or MPL file not found on disk or DDT was built with errors and is not useable.

215000626 CM50_HIS_DDT DDT not History type
Named DDT was built for Input or Output, or MPL history call attempted for a list containing multiple data types.

215000636 CM50_HIS_FAIL Unable to access LCN
History Module has failed, or CG or communications link has failed, or CM50_PKT_PROCESSOR has been aborted.

215000642 CM50_HIS_BEGIN Invalid BEGIN Offset
Invalid Begin_Offset for relative History call.

215000651 CM50_HIS_PART Request complete with some data errors
One or more of the requested data elements had an error on the LCN, see the Status table entries to identify the specific problems.

215000666 CM50_HIS_END Invalid END Offset
Invalid End_offset for relative history call.

215000676 CM50_HIS_TYPE Requested History Type Not Available
History type argument is not a valid code number. Through TDC release 300, History type ranges from 0 to 5.

215000682 CM50_HIS_TAG Requested Tag is Not Historized
The identified point.parameter is not in any History Group

215000692 CM50_HIS_HM Unable to access History Module
History node is not configured on the LCN or has failed.

215000716 CM50_HIS_TIME Request Timed Out -- retry
History module remained busy servicing other requests.

215000722 CM50_HIS_QUEUE History Request Queue id full -- retry
A maximum of 4 History requests can be pending concurrently.

215000732 CM50_HIS_OFFSET Invalid Begin/End Offsets

215000746 CM50_HIS_ARRAY Array Size is too Small
Values array is not large enough to hold the returned number of history values. The arrays should be dimensioned to hold (number_of_values times number of points in DDT) values.

215000756 CM50_HIS_STATE ACP not in RUN state
The ACIDP named in the call (or connected to the calling ACP if not named in the call) is not in RUN state because its ACP is not active or the ACIDP is concurrently.processing another data access request, or DDT for a Get data request was built for Output or History.

215000762 CM50_HIS_ACP ACIDP not found
(Obsolete error code, ACIDP no longer required for History calls)

215000802 CM50_LAX_MONV Invalid Month in VAX date

215000812 CM50_LAX_MONL Invalid Month in LCN date

215000826 CM50_LAX_WRITV Numeric to ASCII conversion error
(Obsolete error code)

215000836 CM50_LAX_READV ASCII to Numeric conversion error
Date/time argument contains a non-numeric character.

215000842 CM50_LAX_NVAL Invalid Number of Values requested
Number of history values must be between 1 and 262. Note: The Number_of_Values argument should be reinitialized before each call; a previous, unsuccessful call could change this variable to zero.

215000852 CM50_LAX_UNIT Invalid History Units specified
For shift or user-averages, the seconds_in_units argument does not allow for a valid calculation of the time span to be retrieved.

215000866 CM50_LAX_MATH Binary Math error
Valid Begin and End times could not be calculated from the specified arguments.

215000876 CM50_LAX_CONA VMS to LCN ASCII conversion error
Invalid format for a VMS date/time variable.

215000882 CM50_LAX_ASCII ASCII to binary conversion error
Invalid ASCII Date/Time value.

215000892 CM50_LAX_BINARY Binary to ASCII conversion error
Binary Time value cannot be converted to a valid Date/time.

215000906 CM50_LAX_ARGRNG Argument out of Range
cg_port_num must be between 1 and 4, or priority must be 1 or 2.

215000916 CM50_LAX_MONTHR Month out of Range
Month portion of date must be between 1 and 12.

215000922 CM50_LAX_BADOFF Invalid History Offset
Negative offsets are not allowed in calls that specify a common_start_time.

215000932 CM50_LAX_HISTYP Invalid History Type

215000946 CM50_LAX_TAG Invalid external tagname format
Tag cannot be parsed into Point.Parameter(index).

215001002 CM50_FIL_MISS Unable to access disk file
Could not open a required file on disk. For ACP operations, the CM50$CONTROL:ACP.TABLE is unavailable, so the change to the ACP Status table could not be checkpointed.

215001012 CM50_FIL_OP Invalid FileAccess Operation
Unable to access a DDT internal file. Make sure that the logical directory CM50$DDT is properly defined.

215001026 CM50_FIL_TYP Invalid FileAccess File type
(Obsolete error message)

215001036 CM50_FIL_II Attempt to write to a .II file
The .II extension is reserved for DDTs, it cannot be used when storing an MPL to disk.
215001042 CM50_FIL_DDTBL DDT not found in system DDT table
Named DDT is not installed, or the CM50$CONTROL: TABLE_NAMES.NM file has been corrupted.
215001052 CM50_FIL_SLEEP Unable to wake CM50_FileAccess
(obsolete error code)
215001316 CM50_FIL_OPER Invalid FileAccess Operation
Another user is holding the DDT Names Table file open, blocking concurrent DDT builds and deletes.
215001322 CM50_FIL_TYPE Invalid File Type
Requested file is not a valid MPL or DDT file. Check the file name.
215001332 CM50_FIL_NDDT Invalid DDT Number
Requests for DDTs by number must be between 1 and the maximum number of DDTs installed (not to exceed 20000)
215001346 CM50_FIL_UNIT Low level communication error:
(FORTRAN) Invalid File Unit used to access a file or File unit already assigned to another file.
215001356 CM50_FIL_CLOS File is already Closed
Request to close a file that is not currently open.
215001402 CM50_FIL_DDT DDT file not found
The referenced DDT is not found on disk. For a Delete request, this means either that the DDT has been deleted previously or its name is misspelled. For other requests, either a file was erroneously deleted or the logical CM50$DDT is not pointing to the correct directory.
215001492 CM50_FIL_NAM DDT name doesn't match source file name
(obsolete error code -- this is now permitted)
215001506 CM50_DDT_NONAM DDT name must be specified
215001516 CM50_DDT_TIME DDT build request timed out
The LCN data access did not return information on all points in the DDT or MPL definition.
215001522 CM50_DDT_SRCERR DDT built with Source Errors
read the error file: CM50$DDT:ddtname.ER for details.
215001532 CM50_DDT_COM No response from communication handler
The CM50_PKT_PROCESSOR has been aborted.
215001546 CM50_DDT_OPEN File open failure
Unable to open an internal DDT or MPL file. Check the requested file name.
215001556 CM50_DDT_READ Error in reading system DDT file
Most likely, the requested MPL file does not have the correct format. Check the file name in the IDB argument.
215001562 CM50_DDT_WRIT File update failed for DDT_IN_CIU file
Improper file protections or disk corruption in the CM50$CONTROL directory.
215001571 CM50_DDT_DELE Delete attempt failed for binary file
File has already been deleted, or VMS file protections are set to block deletion of DDT or MPL file.
215001586 CM50_DDT_SOURCE Unable to find DDT source file
Check specified file path and protections.
215001596 CM50_DDT_USE DDT is currently in use
Only one user/operation may alter a DDT or use it for data access at a time.

215001602 CM50_DDT_DISAR DDT is in disarray and must be rebuilt
DDT build operation had been aborted in midstream.
215001612 CM50_DDT_SRCFIL Error in reading DDT source file
Specified source file does not have the proper format or file access protection.
215001626 CM50_DDT_CGFUL CG Resident Table is full
A maximum of 40 DDTs may be installed as resident in each CG.
215001636 CM50_DDT_CGMEM Insufficient CG memory space available
215001642 CM50_DDT_CGDUP DDT is already CG resident
Redundant request to install a DDT as CG resident.
215001652 CM50_DDT_NAMLEN DDT name cannot exceed 9 characters
215001666 CM50_DDT_EXIST DDT is already built
To modify an existing DDT the REBUILD flag or command option must be set.
215001676 CM50_DDT_QUEUE DDT processing queue is full
A CG can process a maximum of 10 DDT data requests concurrently.
215001682 CM50_DDT_INSTAL DDT is built, not CG resident
215001692 CM50_DDT_INCOM DDT is incomplete
The DDT build request had returned errors. The DDT cannot be used for data transfer until those errors are corrected.
215001706 CM50_ACP_PATH ACP program file not found
The ACP needs to be reinstalled, specifying the correct full pathname of the executable program file. If no directory was specified when the ACP was installed, CM50S tries to read the .EXE file for the named ACP from the CM50$ACP logical directory.
215001716 CM50_ACP_USED ACIDP is connected to another ACP
Connect the ACP to a different ACIDP or disconnect the ACIDP from the other ACP before proceeding.
215001722 CM50_ACP_USE ACP entry is locked by another user
Concurrent use of an installed ACP by multiple processes is not permitted.
215001732 CM50_ACP_LOCK ACP table is locked by another user
Multiple users are attempting to add or delete entries from the ACP Status Table concurrently.
215001746 CM50_ACP_ARG Invalid ACP installation argument
215001756 CM50_ACP_MISS ACP is not installed
The requested ACP name is not found in the ACP status table.
215001762 CM50_ACP_DUP ACP name duplicates another ACP entry
Each ACP must be assigned a unique name when it is installed.
215001772 CM50_ACP_FUL ACP-ACIDP table is Full
The maximum number of ACPs (1000) has already been installed, or the VMS disk quotas/file protection is preventing the addition of a new entry in the CM50$CONTROL:ACP.TABLE file.
215001786 CM50_ACP_EXEC ACP is currently Executing
The installed options of an ACP cannot be changed while it is running. If a change is needed, DEACTIVATE the ACP first.
215001796 CM50_ACP_MODE Invalid program installation mode
ACP installation mode must be 'N','R', or 'T'.
215001802 CM50_ACP_LIST Invalid number of ACPs requested
The status of a maximum of 400 ACPs can be retrieved in one list, and the highest ACP index cannot exceed 1000.
215001812 CM50_ACP_CONECT ACP-ACIDP connection attempt failed
Named ACIDP is not in the specified CG or the ACP is not installed.

215001826 CM50_ACP_ACTIV ACP activation attempt failed
The dispatcher was unable to spawn a process for the ACP. Try activating the ACP as Remote from the ACPOPER screen to get the specific VMS error code. Possible causes include not having the CM50 logical names available system-wide or incorrect setting of VMS privileges or quotas.

215001836 CM50_ACP_DEACT Deactivation attempt failed
The ACP was not actually running in the VAX or the requestor does not have sufficient privileges to abort the ACP process.

215001843 CM50_ACP_NORUN ACP was not in run state
For data transfers, the ACIDP must be in run state. Note: a data access request temporarily changes the state to 'ACCESS' so concurrent data requests through a single ACIDP are blocked.

215001852 CM50_ACP_PID PID is locked by another user
Multiple processes on the VAX cannot use the same process name or PID number.

215001866 CM50_ACP_NOPT ACIDP must be named
For data stores, the ACIDP name is not optional.

215001876 CM50_ACP_DETACH ACIDP is restricted
Remote spawning of ACP and data stores requires it to be installed in Normal mode.

215001882 CM50_ACP_INDX Start_Index for ACP is out of range
ACP index must be between 1 and the number of ACPs installed (never exceeding 1000).

215001892 CM50_ACP_NAME Invalid ACP name
ACP names must be legal VMS process names.

215001906 CM50_ACP_PRIOR Invalid priority for an ACP
The VMS priority for an ACP must be between 0 & 30.

215001916 CM50_ACP_DISC_FAIL ACP/ACIDP disconnect failed
Communications with the CG has failed.

215001922 CM50_DDT_DISC_FAIL DDT/ACIDP disconnect failed
Communications with the CG has failed or the DDT was not connected to the named ACP/ACIDP combination.

215001932 CM50_ACP_CONN ACP already connected to ACIDP
Either the ACIDP or the ACP already has a connection.

215001946 CM50_ACP_PRIV User not authorized for required privilege
See system administrator to ensure that your user registration includes the appropriate privileges and rights identifiers.

215001956 CM50_ACP_UIC No UIC found for named user
Either an attempt to install or execute an ACP using an account name not registered as a VMS user or the user is not authorized to use other account.

215002012 CM50_COM_ACPI ACP terminated by external request
ACP was Deactivated by an operator or DCL STOP/ID was issued for the ACP.

215002116 CM50_COM_SIZE Low level communication error
Invalid arguments in internal CGPIO call.

215002122 CM50_COM_FUNC Invalid CGPIO function code
Invalid arguments in internal CGPIO call.

215002132 CM50_TBL_PRSW Invalid Print Switch value
Invalid arguments in internal CGPIO call.

215002146 CM50_TBL_CGST Unable to Map CG status table
CM50S shared images are not installed.

215002156 CM50_TBL_ACP ACP Table is full
The maximum number of permitted ACPs (1000) has already been installed or the maximum number of concurrent ACP connections are in use by active processes.
215002162 CM50_TBL_PID Error in obtaining a Process Id
Internal error in communications processing.
215002172 CM50_TBL_CG Invalid CG Port Number
Request for a CG port number that is not configured for this system.
215002186 CM50_COM_PACK Error Detected by Packet Processor
Internal error in communications processing.
215002196 CM50_COM_SYS VMS system service call failed
Internal error in communications processing may be the result of starting CM50S from an unprivileged account or a user calling CM50S functions without having the VMS "SYSLCK" privilege active.
215002202 CM50_COM_VAX VAX/VMS system error
215002236 CM50_COM_CG CG system error
215002316 CM50_COM_CTIME Message Confirmation Time Out
CG did not confirm the receipt of a transaction within the configured time out period.
215002322 CM50_COM_RTIME Message Response Time Out
VAX did not receive a data buffer from the CG in response to a request within the configured time out period.
215002332 CM50_COM_ETIME End Message Time Out

215002346 CM50_COM_BUFF Buffer too Small
Insufficient memory allocated for the data buffer.
215002356 CM50_COM_REST CG Restart in Progress
Data cannot be transferred until synchronization of CM50S and CG databases is complete.
215002362 CM50_COM_NULL Null error
Suspect a communications hardware error.
215002372 CM50_COM_WAIT Wait for response
215002386 CM50_COM_SEG Communications Packet Segmentation error
Low level communications error.
215002986 CM50_COM_UNK Unidentified Communications error
215003116 CM50_TX_NS Non-supported Transaction code
CM50 is attempting a function which is not supported by the TDC release currently on the LCN. Most likely, an attempt has been made to access a new release 400 feature on TDC release 300 or earlier.
215003122 CM50_TX_UNK Invalid Transaction code
CGPIO was called using specifying an undefined Transaction.
215003202 CM50_PTR_START Invalid Start Pointer
Starting Offset/pointer is out of range.
215003212 CM50_PTR_END Invalid End Pointer
Ending Offset/pointer is out of range.
215003226 CM50_PTR_SEQ End pointer Greater than Start pointer
215003236 CM50_PTR_RANG Invalid Range
215003242 CM50_PTR_INC Invalid Increment pointer
215003316 CM50_PTR_HIST Invalid History Type
History type argument does not specify a value type retrievable from the History module.
215003322 CM50_PTR_LIST Invalid List Indicator
Point list argument is in error or number_of_values specified for a Point List call exceeds the number of points defined in the Point list array.

215003402 CM50_DDT_START Invalid Start Index
Starting Offset/pointer is out of range.
215003412 CM50_DDT_END Invalid End Index
Ending Offset/pointer is out of range.
215003426 CM50_DDT_SEQ End index Greater than Start index
215003516 CM50_LAX_YEAR Invalid Year
Check format of Date argument.
215003522 CM50_LAX_MONTH Invalid Month
Check format of Date argument. Depending on the call, the month component must be either a number between 01 and 12, or a 3-character (upper case) abbreviation.
215003532 CM50_LAX_DAY Day must be between 1 & 31
215003546 CM50_LAX_HOUR Hour must be between 0 & 23
215003556 CM50_LAX_MINUTE Minute must be between 0 & 59
215003562 CM50_LAX_SECOND Second must be between 0 & 59
215003986 CM50_PTR_UNK Unidentified Format error
CM50S could not interpret the transaction buffer contents.
215004012 CM50_FTF_FILMGR File Manager error
Check mgr_status (Appendix A.4 for details)
215004026 CM50_FTF_BUSY File Transfer utility is busy
Another File Transfer session is in progress
215004036 CM50_FTF_XFRHDR File Transfer internal error
error in buffer header -- call Honeywell TAC
215004042 CM50_FTF_EXISTS File name already in use
215004052 CM50_FTF_MISMATCH File Transfer internal error
error in buffer contents -- call Honeywell TAC
215004066 CM50_FTF_BFRSIZ File Transfer internal error
error in buffer size -- call Honeywell TAC
215004076 CM50_FTF_DATA File Transfer data exception error
215004082 CM50_FTF_RECSIZ File Transfer internal error
error in record size -- call Honeywell TAC
215004092 CM50_FTF_RECNBR File Transfer internal error
error in internal record number -- call Honeywell TAC
215004106 CM50_FTF_READ File Transfer read error
215004116 CM50_FTF_FILNME CG detected Invalid file name
path name not valid on the LCN
215004122 CM50_FTF_ABORT Requested Abort is complete
Transfer was aborted by user Abort File Transfer request
215004132 CM50_FTF_WRITE File Transfer write error
215004146 CM50_FTF_UTILITY Utility Manager error
Check the MGR_STATUS value (Appendix A.4) for details
215004156 CM50_FTF_DATAOUT Data Out file has not been established
Use the Change Dataout Status function
215006562 CM50_FTF_MISARG Missing argument
215006572 CM50_FTF_INVFUNC Invalid function code
215006586 CM50_FTF_INVCG Invalid CG Identifier
CG port number must be 1 to 4 and must be connected to an LCN running release 400 or later.
215006596 CM50_FTF_INVFC Invalid file code
215006602 CM50_FTF_INVDOF Invalid data out function
DO_FUNC argument must be 0, 1 or 2
215006612 CM50_FTF_INVDFC Invalid directory function
DIR_FUNC argument must be 0 or 1

215006626 CM50_FTF_ATTR_ACC Attribute File access error
Attribute file is missing or corrupted. The search for the LCN Attribute file is based on the VAX source file name. Only one or two character LCN suffixes following the last underbar and preceding the ".LA;1" VAX file extension are supported.

215006636 CM50_FTF_ATTR_REV File revision mismatch with Attributes
Data file has been revised so that is does not match the attributes file

215006642 CM50_FTF_ATTR_VER File Version mismatch with Attributes
Version numbers of the data and attribute file on the VAX do not match

215006652 CM50_FTF_SRC Source file OPEN error
Could not open the source file (path name or protections)

215006666 CM50_FTF_CONFIG CM50$CONTROL:FTF_CONFIG.DAT access error
The file identifying extensions for modifiable file extensions is missing or corrupted.

215006676 CM50_FTF_TYPE Invalid record type detected
format of data record does not match type specified by the attribute file

215006682 CM50_FTF_FFB File Transfer internal error
error in file block -- call Honeywell TAC

215006692 CM50_FTF_RCVHDR File Transfer internal error
error in header contents -- call Honeywell TAC

215006706 CM50_FTF_DSTO Destination file open error
could not open requested file (VAX pathname, insufficient disk space or protections)

215006716 CM50_FTF_ATTO Attribute file already exists
use a different name for the destination file

215006722 CM50_FTF_ATTRO Attribute file open error
could not open the attribute file (VAX pathname, insufficient disk space or protections)

215007996 CM50_FTF_UNK_FIL Unable to transfer file
CM50S communications failure

A.3 CM50S SYSTEM STATUS MESSAGES

Numerous diagnostic messages are generated by the CM50S link-level software that are not returned to ACPs as return status codes. These messages are displayed on the CM50S operator console and stored in the VMS operator log file if logging is enabled. These messages are displayed in the format

product - severity - task name - CG number
information further detailing the error

- The product is CM50S.
- Severity is a single character code: (I)nformational, (W)arning, (S)evere or (F)atal.
- The task name describes the ACP or CM50S module reporting the condition.
- The CG number is the number of the CG data link affected

Message descriptions below use the following notation:

XXXX—text that is dependent on the message.
0000—a number which is dependent on the message.
CG#n—will appear as CG#1, CG#2.

A.3.1 Informational Messages

(I)nformational messages announce an event which is normal, but may be of interest. No user interaction is required.

Message: CM50S-I-CM50_PKT_PROC CG#n
LINK ACTIVE
Description: The HDLC controller has detected that the link became active after startup.

Message: CM50S-I-CM50_PKT_PROC CG#n
LINK REINITIALIZATION COMPLETE
Description: The HDLC controller has reinitialized the communication link.

Message: CM50S-I-CM50_PKT_PROC CG#n
STATION ACTIVE
Description: The HDLC controller detected that the station became active after startup.

Message: CM50S-I-CM50_PKT_PROC
PACKET PROCESSOR EXITING
Description: The link-level software is exiting because of a reload or restart of CM50S software.

Message: CM50S-I-CM50_PKT_PROC CG#n
PROCESS DID NOT ABORT
PROCESS DELETED
Description: An abnormal event occurred causing CM50S to abort the current request. A previous error message should have been logged detailing the problem. This message indicates that during message abort, an ACP did not terminate as expected and was stopped by CM50S.

Message: CM50S-I-CM50_PKT_PROC CG#n ROTATE WINDOW UNBLOCKING
Description: Internal package information only; no action is required.

A.3.2 Warning Messages

(W)arning messages indicate an abnormal condition has been detected but normal CM50S operation can continue. No user interaction is required, however, messages may be lost for one or more ACPs awaiting data from the CG link. One abnormal condition may result in a series of warning messages being generated. A high frequency of abnormal condition reporting should be investigated as a possible indicator of CM50 hardware failure.

Message: CM50S-W-CM50_PKT_PROC
UNSOLICITED MSG TIMEOUT -
XXXXXXXXXX MSG DISCARDED
Description: An unsolicited message was received from the CG, but no program requested it before the timer expired.

Message: CM50S-W-CM50_PKT_PROC CG#n
RCVD INVALID XXXXXXXXXX
REQUEST STATE: XXXX
Description: A message was received when it was not expected for the current state.

Message: CM50S-W-CM50_PKT_PROC CG#n
RECEIVED NEGATIVE CONFIRMATION
FOR ACP 00000000
Description: The CG sent a NAK indicating the last message was received in error. The message is automatically retransmitted.

Message: CM50S-W-CM50_PKT_PROC CG#n
ACP NOT FOUND FOR HOST TASK ID: 0000
Description: A message was received, but the ACP was not available to accept it. The ACP has aborted or has been stopped.

Message: CM50S-W-CM50_PKT_PROC CG#n
ACP 00000000 - RCVD INVALID
XXXXXXXXXX MSG STATE: XXXX
Description: The first block of a message was received, but the requesting ACP is not ready to receive it.

Message: CM50S-W-CM50_PKT_PROC CG#n
INVALID MSG RECEIVED FOR ACP 00000000
- TRANSACTION CODE ERROR IN
BLOCK 00 XXXXXXXX
Description: A message was received but its structure is inconsistent. Other associated error messages should give additional information regarding the inconsistency.

Message: CM50S-W-CM50_PKT_PROC CG#n
INVALID MSG RECEIVED FOR ACP 00000000
- NUMBER OF WORDS FIELD OUT OF
RANGE IN BLOCK 00 : 0000

Description: The number of blocks in the received message is not within the range specified by HDLC protocol.

Message: CM50S-W-CM50_PKT_PROC CG#n
INVALID MSG RECEIVED FOR ACP 00000000
- "BLOCKS IN MSG"
ERROR IN BLOCK 00 : 0000
Description: The message received contained a block number that is inconsistent.

Message: CM50S-W-CM50_PKT_PROC CG#n
INVALID MSG RECEIVED FOR ACP 00000000
- BLOCK NUMBER ERROR IN
BLOCK 00 QUE ENTRY 00 : 0000
Description: The blocks of a multiblock message were received out of sequence.

Message: CM50S-W-CM50_PKT_PROC CG#n
INVALID MSG RECEIVED FOR ACP 00000000
- CGID ERROR IN BLOCK 00 : 0000
Description: The cg field of the received message did not contain the same cg number in all blocks of the message.

Message: CM50S-W-CM50_PKT_PROC CG#n
INVALID MSG RECEIVED FOR ACP 00000000
-HOST TASK ID ERROR IN BLOCK 00 : 0000
Description: The host task id of the received message did not contain the same number in all blocks of the multiblock message.

Message: CM50S-W-CM50_PKT_PROC CG#n
INVALID MSG RECEIVED FOR ACP 00000000
- MESSAGE TAG ERROR IN BLOCK 00 : 0000
Description: The message sequence number of the received message did not contain the same number in all blocks.

Message: CM50S-W-CM50_PKT_PROC CG#n
XXXXXXXX COMMAND REJECTED
Description: The HDLC controller rejected a command. This error should be reported to CM50S support personnel.

Message: CM50S-W-XXXXXXXX CG#n
PKT DID NOT CLEANUP THE CGST
Description: An ACP terminated abnormally leaving corrupted internal data structures. The data structures are repaired on the next ACP message request, so no further action is required.

Message: CM50S-W-XXXXXXXX CG#n
CGST NOT CLEANED-UP
Description: An ACP terminated abnormally leaving corrupted internal data structures. CM50S attempts to correct the situation and should report the offending reason.
NOTE: The terminated ACP should be identified and fixed. The ACP that terminated abnormally and caused the problem is NOT the ACP identified in this message. The XXXXXXX in this message refers to the next ACP that attempted to send a message and encountered the corrupted data structures.

Message: CM50S-W-XXXXXXXX CG#n
PARTIAL CGST CLEANED-UP
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
ACP DELETION DURING SEGMENTATION
Description: The following five messages indicate possible reasons for the internal data structure corruption errors described above.

Message: CM50S-W-XXXXXXXX CG#n
ACP DELETED BEFORE START OF TRANSMISSION
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
ACP DELETED DURING TRANSMISSION
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
ACP DELETED DURING WAIT FOR CONFIRMATION
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
ACP DELETED AFTER MSG CONFIRMATION
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
Unknown ACP termination state
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
CGPIO BUFFER TOO SMALL FOR INCOMING
DATA: BUFSIZE = 0000 WORDS
Description: The buffer size argument passed to the link-level software was not large enough to contain the message received.

Message: CM50S-W-XXXXXXXX CG#n
UNABLE TO PRINT MESSAGE -
ERROR LOCKING DUMP FILE
ERROR %X 00000000
Description: The diagnostic dump file could not be locked to dump a message. It can be locked by a user viewing it with an editor.

Message: CM50S-W-XXXXXXXX CG#n
UNABLE TO PRINT MESSAGE - UNABLE TO OPEN DUMP MSG FILE,
STATUS = %X' 00000000
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
UNABLE TO PRINT MESSAGE - ERROR UNLOCKING DUMP FILE'
ERROR 00000000
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
ERROR MAPPING TO CGST TABLE
Description: The XXXXXXX ACP was aborted because of a VMS system service failure. The failure code was reported back to the ACP. Clues to the nature of the failure can be found using the error codes returned to the ACP.

Message: CM50S-W-XXXXXXXX CG#n
ERROR REQUESTING ACP PID
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
ERROR LOCKING TABLE NAME: XXXXXXXX
ERROR %X' 00000000
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
ERROR UNLOCKING TABLE NAME: XXXXXXXX
ERROR %X' 00000000
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
ERROR LOCKING CGST_CURRENT_HTID
ERROR %X' 00000000
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
ERROR UNLOCKING CGST_CURRENT_HTID
ERROR %X' 00000000
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
ERROR LOCKING TABLE NAME: XXXXXXXX
ERROR %X' 00000000
Description: See above.

Message: CM50S-W-XXXXXXXX CG#n
ERROR UNLOCKING TABLE NAME: XXXXXXXX
ERROR %X' 00000000
Description: See above.

Message: CM50S-W-XXXXXX
NON-ZERO FORWARD LINK
Description: An internal problem has been detected in the CM50S data structures. This warning is always followed by a fatal packet processor message. This error should be reported to CM50 support personnel.

Message: CM50S-W-XXXXXX
NON-ZERO BACKWARD LINK
Description: See above.

Message: CM50S-W-XXXXXX
ERROR APPENDING PACKET TO QUEUE
ERROR - %X 00000000
Description: See above.

Message: CM50S-W-XXXXXX
ERROR REMOVING PACKET FROM QUEUE
ERROR - %X 00000000
Description: See above.

Message: CM50S-W-XXXXXX
ATTEMPT TO REMOVE A PACKET FROM AN
EMPTY QUEUE
Description: See above.

Message: CM50S-W-XXXXXX
ERROR WAKING UP PACKET PROCESSOR
PROCESS ERROR - %X 00000000
Description: See above.

A.3.3 Severe Messages

A (S)evere message indicates that a serious problem has been detected impacting the performance of the indicated CG link. In an attempt to recover the link, reinitialization is automatically initiated. All ACPs awaiting data from the affected CG are aborted and must be restarted.

Message: CM50S-S-CM50_PKT_PROC CG#n
ACP 00000000 - MESSAGE TRANSMIT
TIMEOUT
Description: The message to be transmitted for ACP with process id 00000000 was not transmitted within the specified time. The link is probably down.

Message: CM50S-S-CM50_PKT_PROC CG#n
ACP 00000000 - MESSAGE CONFIRM
TIMEOUT
Description: Message confirmation for the last message sent by process 0000000 was not received within the specified time.

Message: CM50S-S-CM50_PKT_PROC CG#n
ACP 00000000 - RESPONSE TIMEOUT
Description: The response to the last message sent by process 0000000 was not received within the specified time.

Message: CM50S-S-CM50_PKT_PROC CG#n
ACP 00000000 - END OF MESSAGE TIMEOUT
Description: The last block of a multiblock message response was not received within the specified time.

Message: CM50S-S-CM50_PKT_PROC CG#n
OUTGOING BLOCK NUMBER OUT OF SEQUENCE
Lstblk - 0000
Curblk - 0000
Lstnum blk - 0000
Curnum blk - 0000
Lst htid - 0000
Cur htid - 0000
Lst ACP - 0000
Cur ACP - 0000
Description: This message provides diagnostic information when an outgoing message contains conflicting information.

Message: CM50S-S-CM50_PKT_PROC CG#n
OUTGOING NUMBER BLOCKS IS INCORRECT
Lstblk - 0000
Curblk - 0000
Lstnum blk - 0000
Curnum blk - 0000
Lst htid - 0000
Cur htid - 0000
Lst ACP - 0000
Cur ACP - 0000
Description: See above.

Message: CM50S-S-CM50_PKT_PROC CG#n
OUTGOING NUMBER WORDS IS INCORRECT
Num words - 0000
Lstblk - 0000
Curblk - 0000
Lstnum blk - 0000
Curnum blk - 0000
Lst htid - 0000
Cur htid - 0000
Lst ACP - 0000
Cur ACP - 0000
Description: See above.

Message: CM50S-S-CM50_PKT_PROC CG#n
OUTGOING DATA SIZE IS INCORRECT
Data size - 0000
Num words - 0000
Lstblk - 0000
Curblk - 0000
Lstnum blk - 0000
Curnum blk - 0000
Lst htid - 0000
Cur htid - 0000
Lst ACP - 0000
Cur ACP - 0000
Description: See above.

Message: CM50S-S-CM50_PKT_PROC CG#n
RETRY LIMIT EXCEEDED
Description: CM50S has attempted to retransmit a message, but has exceeded the retry count.

Message: CM50S-S-CM50_PKT_PROC CG#n
FLAG TIMEOUT
Description: The HDLC controller has detected that flag synchronization has been lost. The link is probably down.

Message: CM50S-W-CM50_PKT_PROC CG#n
UNKNOWN ICP EXCEPTION PACKET RECEIVED
CONTROL CODE - 00000000
Description: An unknown message has been received from the HDLC controller. This warning should be reported to CM50 support personnel.

Message: CM50S-S-CM50_PKT_PROC CG#n
LINK INACTIVE
Description: The HDLC controller has detected that the physical link is inactive. The link is probably disconnected.

Message: CM50S-S-CM50_PKT_PROC CG#n
STATION INACTIVE
Description: The HDLC controller has determined that the station is inactive. The link is probably down.

Message: CM50S-S-CM50_PKT_PROC CG#n
STATION HAS BEEN RESET
Description: The HDLC station has been reset at a time other than startup. This message should be reported to CM50 support personnel.

Message: CM50S-S-CM50_PKT_PROC CG#n
LINK REINITIALIZATION IN PROGRESS
Description: The link-level software is reinitializing the HDLC link. This is usually caused by a previously reported severe condition.

A.3.4 Fatal Messages

A (F)atal message indicates that a very serious problem has been detected and must be investigated. A fatal error message is generated only as the result of a nonrecoverable error being detected in the CM50S software, the VMS operating system, or the CM50 hardware. CM50S software is left in a nonfunctional state and must be stopped and restarted. Any time a fatal error message is displayed these steps should be followed:

(1) Examine the CM50S operator console and/or the VMS system operator log file for possible information relating to the problem. If a VMS system operator log file is available, copy it to the directory CM50$LOG.

(2) Save the entire contents of the directory CM50$LOG for later analysis by CM50S support personnel.

(3) Contact CM50S support personnel, if possible, before attempting to stop and restart CM50S.

Message: CM50S-F-CM50_PKT_PROC
ERROR READING ICP XXXX - %X 0000000

Message: CM50S-F-CM50_PKT_PROC
PACKET RECEIVED WITH INVALID LINK ID.
LINK ID: 0000
STATION ID: 0000

Message: CM50S-F-CM50_PKT_PROC CG#n
ERROR POSTING READ TO ICP XXXX - %X 00000000

Message: CM50S-F-CM50_PKT_PROC
ERROR IN WRITE TO ICP XXXX - %X 00000000

Message: CM50S-F-CM50_PKT_PROC CG#n
ERROR RESETING ICP XXXX - %X 00000000

Message: CM50S-F-CM50_PKT_PROC CG#n
IOSB ERROR RESETING ICP %X 00000000

Message: CM50S-F-CM50_PKT_PROC
PACKET PROCESSOR EXITING -
DUMP FILE CREATED
STATUS - %X 00000000
Description: The low level communication module has exited, caused by a fatal error. A dump file has been created for diagnostic purposes. CM50S must be restarted.

Message: CM50S-F-CM50_PKT_PROC
ICP WRITE OVERLOAD

Message: CM50S-F-CM50_PKT_PROC
ERROR POSTING WRITE TO ICP XXXX %X 00000000

Message: CM50S-F-CM50_PKT_PROC
Error locking table: XXXXXXXX

Message: CM50S-F-CM50_PKT_PROC
Error unlocking table name: XXXXXXXX

Message: CM50S-F-CM50_PKT_PROC
ERROR SETTING 1 SECOND TIMER

Message: CM50S-F-CM50_PKT_PROC
ERROR MAPPING TO CGST TABLE

Message: CM50S-F-CM50_PKT_PROC
ERROR REQUESTING PACKET PROCESSOR PID

Message: CM50S-F-CM50_PKT_PROC CG#n
ERROR ASSIGNING CHANNEL TO ICP

Message: CM50S-F-CM50_PKT_PROC CG#n
ERROR POSTING READ TO ICP XXXX

Message: CM50S-F-CM50_PKT_PROC CG#n
ERROR WAKING UP PROCESS
PID - %X 00000000

A.4 FILE TRANSFER MANAGEMENT STATUS CODES

A.4.1 LCN File Manager Status codes

When an LCN File Transfer request has a return_status of CM50_FTF_FILMGR (215004012) then the specific problem is identified by the mgr_status code:

0	Transfer in progress
1	End of File
2	Timeout expiration
3	Data hard error
4	FAB hard error
5	Directory hard error
6	Inconsistent command
7	Device timeout
8	Invalid command
9	Invalid LRN
10	Open files exhausted
11	LRNs exhausted
12	LRN unassigned
13	File access denied
14	Device access denied
16	Incompatible option
17	Invalid file name
18	Duplicate file name
19	File not found
20	Device not found
21	Access violation
22	Invalid buffer length
26	Insufficient storage space
27	No local request class system memory
28	End of directory
31	Buffer overflow
32	Volume not mounted
34	File in use
35	Volume in use
38	Internal error
43	Size conflict
44	Invalid variable record length
46	File not open
48	Unspecified device error
49	Invalid buffer address
50	LRN not on volume
51	Success
52	Invalid volume name
53	Volume not found
54	Duplicate volume
55	Duplicate volume alias
56	Volume alias not found
57	Logical device not found
58	Duplicate logical device
59	No local request class user memory

60 No remote request class memory
61 Heap manager failure
62 Unimplemented function
63 Attributes incompatible
64 LRN cancelled
65 LRN failed
66 Illegal LRN deallocation
67 LDIDs exhausted
68 Invalid device ID
69 Invalid CRB Identifier
70 Unsupported feature on personality
71 Corrupted directory data
72 VVAT table full
73 Maximum tracks exceeded
74 LRN allocation denied
75 Device failed
76 Request class table error
77 Invalid request class data
78 Volume access denied
79 Invalid physical node identifier
80 Invalid file configuration revision
81 Volume alias not empty
82 Device not redundant
83 Redundant device not available
84 Illegal device state transition
85 Option not purchased
86 Invalid device address
87 Descriptors not found
88 Remote LCN has not connected to local LCN
89 Local LCN has not connected to remote LCN
90 Volume Read permission on remote LCN denied
91 Volume Read/Write permission on remote LCN denied
92 Specified remote LCN not defined
93 Not mutually connected to remote LCN

A.4.2 LCN Utility Manager Status codes

When an LCN File Transfer request has a return_status of CM50_FTF__UTILITY (215004146) then the specific problem is identified by the mgr_status code:

3	Incomplete command
4	Invalid pathname
6	No files on volume
8	Invalid option
9	Invalid command format
21	Extra characters
22	Out of memory
24	Function not implemented
30	Illegal use of wildcard
32	Bad destination file size
33	Destination pathname required
34	Source file error
35	Destination file error
36	Temporary file error
37	Max files out of range
38	Invalid directory
39	Blocksize out of range
40	Destination file extension not allowed
53	Illegal path format for create
56	File manager pointer error
57	Can not rename volume on HM
63	Bad source drive number
67	Bad destination drive number
68	Destination same as source
70	Error in perform PIO
73	Memory allocation error
77	Invalid physical node number
81	Invalid drive number
82	Synchronization was initiated
91	Duplicate volume ID
92	FMD too long
94	Dismount service error
95	Not a local device
97	Physical node nonexistent
98	Cannot file HM nodes on Network
99	No running HM nodes on Network
101	Maximum memory directory files
102	Memory directory no BS needed
103	Device not present
104	Device failed
105	Device offline
106	Volume corrupted
107	Device not formatted
108	Device not mounted
109	Device resource error
110	Device synchronizing
111	Device degraded
112	Device formatting
113	Device state unknown

SYSTEM SOFTWARE ISSUES
Appendix B

This appendix contains information on CG-VAX system-software issues such as CG/PLNM configuration, system startup, tuning considerations, and failure restarts.

CAUTION

Because the CM50S has virtually unlimited access to data points on the LCN, all precautions should be taken to ensure that only qualified personnel have access to ACP installation and modification. There is no substitute for thorough preparation and testing of application programs.

B.1 CONFIGURATION OF A CG OR PLNM

The first step in configuring the CG/PLNM is—from a TDC 3000 Universal Station—to modify the system NCF to include the CG/PLNM node. Refer to *Network Data Entry* as a starting point if this procedure is unfamiliar. As part of LCN Nodes Configuration, you will assign one or more Process Units to the CG (see *Network Form Instructions* and Appendix E of this manual for additional information).

Additional CG/PLNM configuration entries are made through the CG Configuration display. To reach this display, call up the Engineering Main Menu at a Universal Station on the same LCN as the CG/PLNM, then select the "Computing Module" target. This brings up the Computer Gateway Build and Configuration menu from which you select the "CG Configuration" target.

Your selection of CG configuration type is controlled by the hardware used to interface your VAX to the LCN. Select "CNI" if the connection is through a PLNM, and "CLI-HDLC" if connection is through a CG. Additional choices you will make depend on this selection and are explained below.

NOTE

On completion of CG/PLNM configuration (or configuration change), you must demand a CG checkpoint, then shutdown and restore the node before the changes take effect.

The final step in CG/PLNM configuration, ACIDP and CRDP point building (see Section 4 in this manual), can wait until just before you are ready to install an ACP.

B.1.1 PLNM Configuration Choices

The only configuration choice for a PLNM (after selecting "CNI" on the Computer Gateway Configuration display) is "Confirmation Timeout." Normally, you will leave this entry at its default value of 10 seconds.

B.1.2 CG Configuration Choices

There are several configuration choices to be made for a CG (after selecting "CLI-HDLC on the Computer Gateway Configuration display). The following values are normal for CM50S configuration of a CG:

Time Synch Period—15 seconds
Confirm Timeout—2 seconds
Floating Point Conv—IEEE 754
Baud Rate—57600
Link Protocol—HDLC_LAPB
Station Address—CG=1
T1 Time Unit—14
N2 Count—3

B.2 ASSIGNMENT OF CG/PLNM PORTS

All CM50S communications with CG/PLNMs use a port number as an address. These port numbers are assigned to specific Plant Network Modules or Computer Gateway addresses through a configuration utility. This utility should be run during initial CM50S installation and whenever a CG/PLNM is added or its address is changed. Note that replacing the CNI board on a PLNM will change its address.

The CG/PLNM configuration utility can be executed only from the CM50_MGR or other priviledged system account. The display and validation criteria are determined by the communications bus type specified during CM50S installation.

The logical CM50$VER must be defined as the current release (CM50$:[R040]) and CM50$CONFIGURATION must be defined (normally as CM50$:[CONTROL] CONFIGURATION.DAT). The CG/PLNM configuration utility is invoked by typing:

```
RUN CM50$EXE:CM50_CONFIGURE_CGS
```

If CG/PLNM ports previously have been configured, the values from the current file are displayed. The following function keys are active:

PF4/QUIT—Exit the utility. The disk file is not affected. Note that when the display shows "default values," there is no configuration file on the disk.

G0/STORE—Validate and store the currently configuration data to disk. If any errors are detected, an appropriate message is displayed. Nothing is store to disk until any error(s) are corrected and the STORE function is pressed again.

G7/PRINT—Prints the configuration as displayed on the default printer.

PF2/HELP—Displays a brief explanation of the field where the cursor is presently located.

B.2.1 PLNM Ports

The configuration of Plant Network Modules connected to the VAX by the Local Area Terminal (LAT) ethernet protocol uses the display shown below.

CONFIG LAT **CONFIGURATION OF PLANT NETWORK MODULES** 5 OCT 92 14:01

CG PORT	**Ethernet Address**	**Alternate Address**
1: Description	HVN_ **08002B241065**	HVN_ **08002B123999**
	Node 16 on Main Process LCN	
2:	HVN_ **08002B123456**	_____
	Node 14 on Boiler Control LCN	
3:	HVN_ _____	_____
4:	HVN_ _____	_____

PF4	PF2	G7	G9	G0
QUIT	**HELP**	**PRINT**	**ADD_HDLC**	**STORE**

The action required is to specify which primary ethernet addresses should be assigned to each CG port. The CG ports must be assigned sequentially. That is, CG port 2 cannot be assigned unless CG port 1 is also assigned. The Alternate Address is used to assign other PLNMs as failover node The ethernet address (following the "HVN_" prefix) is 12 hexadecimal characters (0-9, A-F) and needs to match the address printed on the CNI board in the PLNM.

- `Description`—Used to define a LAT service broadcast message for each primary PLNM.
- `ADD_HDLC`—Used to configure Computer Gateways connected from HDLC links to the VAX.

B.2.2 CG Ports

The configuration of Computer Gateways connected through HDLC links involves both the addressing of the SIMPAC communications boards and the setting of options for each port as shown below.

CONFIG HDLC **CONFIGURATION OF COMPUTER GATEWAYS** 5 OCT 92 14:04

HDLC BOARD	**1**	**2**
Device:	ZQA0	
CSR:	761000	

CG PORT	**1**	**2**	**3**	**4**
Device:	ZQA0	**08002B123456**		
Link:	2			
Station:	3			
Alt.Device:		**08002B123999**		
Speed:	57600			
Modem (Y/N):	N			
timeout:	2			
retries:	3			

Desc.**1**:
2: Node 14 on Boiler Control LCN
3:
4:

PF4	PF2	G7	G0
QUIT	**HELP**	**PRINT**	**STORE**

CGs can be connected to ports on one or two HDLC boards in the VAX. The installation of these boards is described in Appendix F. The appropriate VAX address (four characters beginning with "Z") needs to be copied as the HDLC BOARD Device. For BI bus systems, the BI slot number must also be copied as the HDLC BOARD Node. (For QBUS and UNIBUS systems, the corresponding CSR address is calculated from the device address, thus does not have to be entered.) The order of the boards (1 and 2) has no significance.

The CG Ports must be assigned sequentially. That is, CG Port 2 cannot be assigned unless CG Port 1 also is assigned. The following fields need to be specified for each CG.

- `Device`—Address of the HDLC board connected to the CG. This must match one of the displayed board device names. To configure a CG port for an ethernet connection to a PLNM, enter the 12-character hexadecimal ethernet address (following the "HVN_" prefix) as the device and leave the remaining fields (`Link` through `retries`) blank.

- `Link`—The number of the port on the HDLC board. The possible range is 0 to 3 for BI bus and QBUS systems, and 0 to 7 for UNIBUS systems. Note that only two links on a board can be used with transmission speeds above 19200. For BI bus and QBUS systems, the high speed links are ports 2 and 3; for UNIBUS systems, the high speed links are ports 6 and 7.

- `Station`—The logical HDLC station number must be either 1 or 3 and cannot be the same as configured on the LCN. Normally the LCN is configured for the CLI board in the CG to be station 1, so the HDLC board in the VAX should be station 3.

- Alternate `Device`—An ethernet address of a PLNM used for failover. This is used only when the primary CG port has been configured from the PLNM configuration menu.

- `Speed`—The baud rate for communications. This should match the baud rate of the CG as configured on the LCN. Legal speeds are 1200, 2400, 4800, 9600, 38400, and 57600. Normally, 57600 is used for high speed links.

- `Modem`—Specifies whether or not the CG is connected by fiber optic modems. "N" indicates a normal (direct cable) connection with the clock provided by the HDLC board. "Y" indicates that an external clock signal is being provided by modems used to extend the permissible distance between the VAX and the CG.

- `timeout`—The maximum wait time (in seconds) for acknowledtement of a transmission before retrying or returning an error. Permitted range is 1 to 255. Normally set to 2.

- `retries`—The number of transmision attempts to make on an outgoing transaction before returning a transmission error to the calling application. Permitted range is 1 to 255. Normally set to 3. Be aware that a retry value of 1 means a single transmission attempt no retry after a communications error.

- `Description`—Used to define a LAT service broadcast message for a PLNM or to describe the HDLC CG ports.

B.3 CM50S SOFTWARE INSTALLATION

Release 4.0 of CM50S uses approximately 10,000 blocks (5.2 Megabytes) of disk space. Execution requires approximately 3800 free global page table entries and four global sections. Recommended non-standard user account quotas are ENQLM = 250 and PGFLQUO = 20000.

CM50S installation requires that the communications be properly configured on both the VAX and the connected CGs.

CM50S uses the standard DEC VMSINSTAL facility for installation. Detailed installation guidelines are incorporated into the CM50S Release Notes for a specific release. Please refer to that documentation for CM50S product installation. Familiarity with the DEC VMSINSTAL facility is helpful but not required.

B.4 CM50S DIRECTORIES AND FILES

CM50S is controlled through a collection of logical names. All logical names, except the distribution root directory pointer CM50$, are defined in the startup command procedure CM50_STARTUP.COM. If the defaults established for any of the logical names are not compatible with the directory structure on a specific system, many of the logical names can be redirected to provide load sharing across multiple disk drives at the system manager's discretion. A brief explanation of the most important logical names follows.

CM50$CONFIGURATION	Hardware configuration data file
CM50$DRIVER	Device driver root directory
CM50$DDT	DDT binary files

The following control the default locations referenced if the desired file is not found in the user's default directory and a complete pathname is not specified.

CM50$DDT_SRC	DDT source files
CM50$ACP	ACP executable files

For all CM50S detached processes, SYS$INPUT, SYS$OUTPUT, and SYS$ERROR are directed to CM50$INPUT, CM50$OUTPUT, and CM50$ERROR.

CM50$INPUT	Console Input
CM50$OUTPUT	Console Output
CM50$ERROR	Error Logger

CM50$OPERATOR determines the operator console (OPER1, OPER2, ... OPER12) to receive CM50S system status messages.

CM50$OPERATOR	Operator Console

CM50S images and data structures

CM50$EXE	CM50S executables and controlling command files
CM50$LIB	User-visible include files and command files
CM50$FORMS	Forms and menus
CM50$SUPPORT	Honeywell support tools
CM50$EXAMPLES	Source code for sample programs
CM50$CONTROL	Control files
CM50_SHARE	Sharable image
CM50_COMM1	Data COMMON
CM50$LOG	Status Log Directory

Examine the file CM50_STARTUP.COM for additional information. For ease of maintenance, the following local symbols, defined at the beginning of CM50_STARTUP, are used in defining various logical names:

LOG_TABLE	A local symbol controlling the table in which logical names are installed.
CONS_OUT	A local symbol controlling the assignment of SYS$OUTPUT for CM50S control programs that run in detached processes.
CONS_INP	A local symbol controlling the assignment of SYS$INPUT for CM50S control programs that run in detached processes. Must be directed to NL: unless otherwise directed by Honeywell TAC.
CONS_ERR	A local symbol controlling the assignment of SYS$ERROR for CM50S control programs that run in detached processes.
ACP_OUT	A local symbol controlling the assignment of SYS$OUTPUT for ACPs running in detached processes.
ACP_INP	A local symbol controlling the assignment of SYS$INPUT for ACPs running in detached processes.
ACP_ERR	A local symbol controlling the assignment of SYS$ERROR for ACPs running in detached processes.
OPER_CONS	A local symbol controlling the destination for operator messages. Must be a number in the range 1 to 12 (denoting OPER1 to OPER12) to designate a specific operator terminal.
VER	A local symbol containing the current release I.D.

CM50S DIRECTORY & FILE SUMMARY

[.ACP]	Suggested directory for installed ACP executable files
[.ACP.SOURCE]	Suggested directory for ACP source files
[.CONTROL]	This directory holds the checkpoint files for the ACP and DDT control structures. Any editing or other DCL-level access of these files will endanger the integrity of CM50S operations.
ACP.DETAIL	Controls the proper activation of ACPs
ACP.TABLE	Checkpoint of the ACP Status Table
ACPI_PTR.DAT	Pointers for the ACP Installation Log
ACPI_REC.DAT	Data for the ACP Installation Log
CM50_SCHED.TBL	Controls scheduled execution of jobs
CONFIGURATION.DAT	Controls the configuration of CG ports
DDT_IN_CIU	Controls the loading of CG-resident DDTs
FTF_CONFIG.DAT	List of LCN extensions treated as modifiable ASCII files by the LCN File Transfer facility.
TBL_NAMES.NM	Checkpoint of the Installed DDT list
[.DDT]	Suggested directory for installed DDT binary files
[.DDT.SOURCE]	Suggested directory for installed DDT source files
[.DRIVER]	HDLC controller drivers directory. A sub-directory is present for each required bus type
[.ERROR_LOG]	System error and diagnostic logs directory
[.R040]	
CM50.DOC	Internal documentation of the release level and communications bus type
CM50S_MGRSTART.COM	To start up CM50S during reboot
CM50_STARTUP.COM	Interactive startup of CM50S operations
CM50_STOP.COM	Orderly shutdown of CM50S operations
[.R040.BLANK_CONTROL]	Blank data tables. The system can be totally reinitialized by copying these files to the CM50$CONTROL directory, replacing the existing files.
[.R040.EXAMPLES]	Miscellaneous system examples directory
[.R040.EXE]	This directory contains the executable images used in CM50S operations. Many of these are system configuration and control utilities. Only those files marked with an asterisk (*) should be run as applications programs.
* ACPOPER.EXE	Interactive ACP operations
BUILD_HDLC_STARTUP	Establishes communications directives
* CGDSP.EXE	Interactive CG database displays
CM50_COMM1.SHR	Global shared data section

*	CM50_CONFIGURE_CGS	Utility to configure CG ports
	CM50_FTF_SHARE.EXE	Global selection for LCN File Transfer
	CM50_SHED.EXE	Detached process for job scheduling
	CM50_SHARE.EXE	Global section of subroutines
	CM50_USS.EXE	Internal User System Services
*	DDTOPER.EXE	Interactive DDT operations
	DELETE.EXE	Part of orderly shutdown
*	FTFXFER.EXE	Interactive File Transfer operations
	IASDMENU.EXE	Menu driver utility
	INITCOM.EXE	Loads Global section from checkpoint files
	LLCM_SHARE.EXE	Global section for communications routines
*	MAKEINC.EXE	Interactive Include file generator
	PKT.EXE	Detached process for communications
	RESTART.EXE	Detached process for CG Link synchronization
	SHUTDOWN.EXE	Part of orderly shutdown
	SPATCH.EXE	Detached process for remote activation of ACPs
	STARTUP.EXE	Initializes communications
*	TEST_LINK.EXE	Interactive utility to confirm communications
[.R040.LIB]		Include files
	CM50.COM	User entry point to CM50S software package
	ACP_COMMAND.CLD	DCL interface
	DDT_COMMAND.CLD	DCL interface
	FTF_COMMAND.CLD	DCL interface to LCN File Transfer
	CM50_ERROR_MSG.EXE	Message Library
	CM50_ERROR_MSG.MSG	Text for Message Library
[.R040.FORMS]		Screen display format files
[.R040.SUPPORT]		Honeywell TAC support tools directory
[.R040.HELP]		Help text files
[.R040.US_IO]		Communications routines used by Honeywell layered products

B.5 RESTART PROCEDURE

Responsibility for establishing communication between the CG and VAX is left to the CG/PLNM. Whenever the CG/PLNM is freshly loaded it attempts to make a Cold Restart. If communications with VAX are interrupted following a successful Cold Restart, the CG/PLNM then attempts a Warm Restart.

In either instance, the major purpose of restart is to resolve any mismatch of databases between the CG and VAX. Upon completion of restart, the VAX displays a restart message at the operator console and the CG/PLNM begins scheduling of ACPs.

If communication is broken at any time, the CG/PLNM waits for two minutes before trying again. Some failures may require reload of the CG/PLNM.

The restart program on the VAX is the detached process CM50_RESTART.

B.6 COMMUNICATIONS TROUBLESHOOTING

There are many conditions that will cause a COMMDOWN status to be shown at the LCN. Following are lists of things to check for HDLC and PLNM connections:

B.6.1 Troubleshooting HDLC Communications

1) CM50S must be running. From a VAX terminal, do a SHOW SYSTEM to confirm that the CM50_PKT_PROC process is active.

2) The Computing Module configuration at the LCN must be set up for HDLC (CM50).

3) The cable between the CG and distribution panel in VAX must connect the right port on back of the CG (JA) with the left port on the distribution panel (channel 2 and single CG).

4) (QBUS HDLC Only) Simpact board:
   - SW0-SW3 must be pinned all 2's
   - Cable in J23 for high speed link (or cable in J01 for low speed link)

5) VAX control files: If you get LINK_INIT begin and complete messages and the VAX shows BEGIN RESTART but the COMMDOWN indication remains, then one of the following files in the CM50$CONTROL direct may be corrupt and should be replaced by copying the file of the same name from the CM50:[R0xx.BLANK_CONTROL] directory:
   - DDT_IN_CIU (list of CG-resident DDTs)
   - TBL_NAMES.NM (list of all installed DDTs)
   - ACP_TABLE (list of all installed ACPs)

6) VAX configuration file CONFIGURATION.DAT
   - use port 2 or 3 for high speed link
   - HDLC protocol calls for a 1 or 3 in this file and the opposite (3 or 1) at the LCN configuration

B.6.2 Troubleshooting PLNM Communications

1) CM50S must be running. From a VAX terminal, do a SHOW SYSTEM to confirm that the CM50_PKT_PROC process is active.

2) The Computing Module configuration at the LCN must be set up for CNI (CM50).

3) The CM50_PKT_PROC process will display messages if it is unable to properly connect fo the LAT service.

   • Verify that the hardware address specified in the configuration file is correct.

   • Verify that LATCP is able to "see" external services. (This is accomplished by executing the command $ MCR LATCP SET NODE/CONNECTIONS = BOTH.)

   • Verify that the CM50_PKT_PROC has enough quota to encompass the TTY_ALTYPAHD and TTY_TYPAHDSZ specified.

B.6.3 Using the LAT Protocol

Local Area Transport (LAT) is a serial transport protocol included with VMS. LAT originally was designed to support communications between a server and a "dumb" device in a local area network. Starting with VMS release 5.4-1, LAT has been enhanced to allow connection-oriented serial transmission between VAX nodes. CM50S uses LAT when it is connected to the LCN through a PLNM.

B.6.3.1 LAT Startup

Addition of the new functionality has required a completely new LAT start-up procedure.

Old sequence: SYS$STARTUP:SYSTARTUP_V5.COM called the file SYS$STARTUP:LTLOAD.COM.

New sequence: SYS$STARTUP:SYSTARTUP_V5.COM calls SYS$STARTUP:LAT$STARTUP.COM which calls SYS$MANAGER:LAT$SYSTARTUP.COM.

CM50S startup requires changes to SYS$MANAGER:LAT$SYSTARTUP.COM and to SYS$STARTUP:SYSTARTUP_V5.COM. Figure B-1 illustrates the flow of the three files and highlights the required changes.

The SYS$STARTUP:SYSTARTUP_V5.COM file requires the node name and an identity string to follow the invocation of SYS$STARTUP:LAT$STARTUP.COM. Normally, the node name is the same as the DECnet node name, and the identity string shows the node name and VMS version. The SYS$MANAGER:LAT$SYSTARTUP.COM.file requires the following line:

```
$lcp set node/connections=both
```

This line in a slightly different form already appears in the template startup file. It needs to be modified to appear as above.

The SYS$STARTUP:SYSTARTUP_V5.COM file needs to have LATCP program initiation removed. LATCP is started in the SYS$STARTUP:LAT$STARTUP.COM file. DEC recommends that this latter file **not** be modified by users.

B.6.3.2 LAT Control Program

The LAT control program is the utility program that you use to configure and control the LAT protocol. Systems that join a LAT configuration are called service nodes. The commands of interest to maintaining and configuring CM50S are:

`$MCR LATCP` or `RUN SYS$SYSTEM:LATCP`—invokes LATCP

`LCP> SHOW SERVICES`—displays all the services on the LAT network. A service is offered by all nodes that are configured to accept incoming connections.

`LCP> SET NODE/CONNECTIONS = INCOMING`—tells this node to accept connection requests from other nodes (incoming connections). Executing the Show Services command will display only this node. This node cannot connect to other nodes.

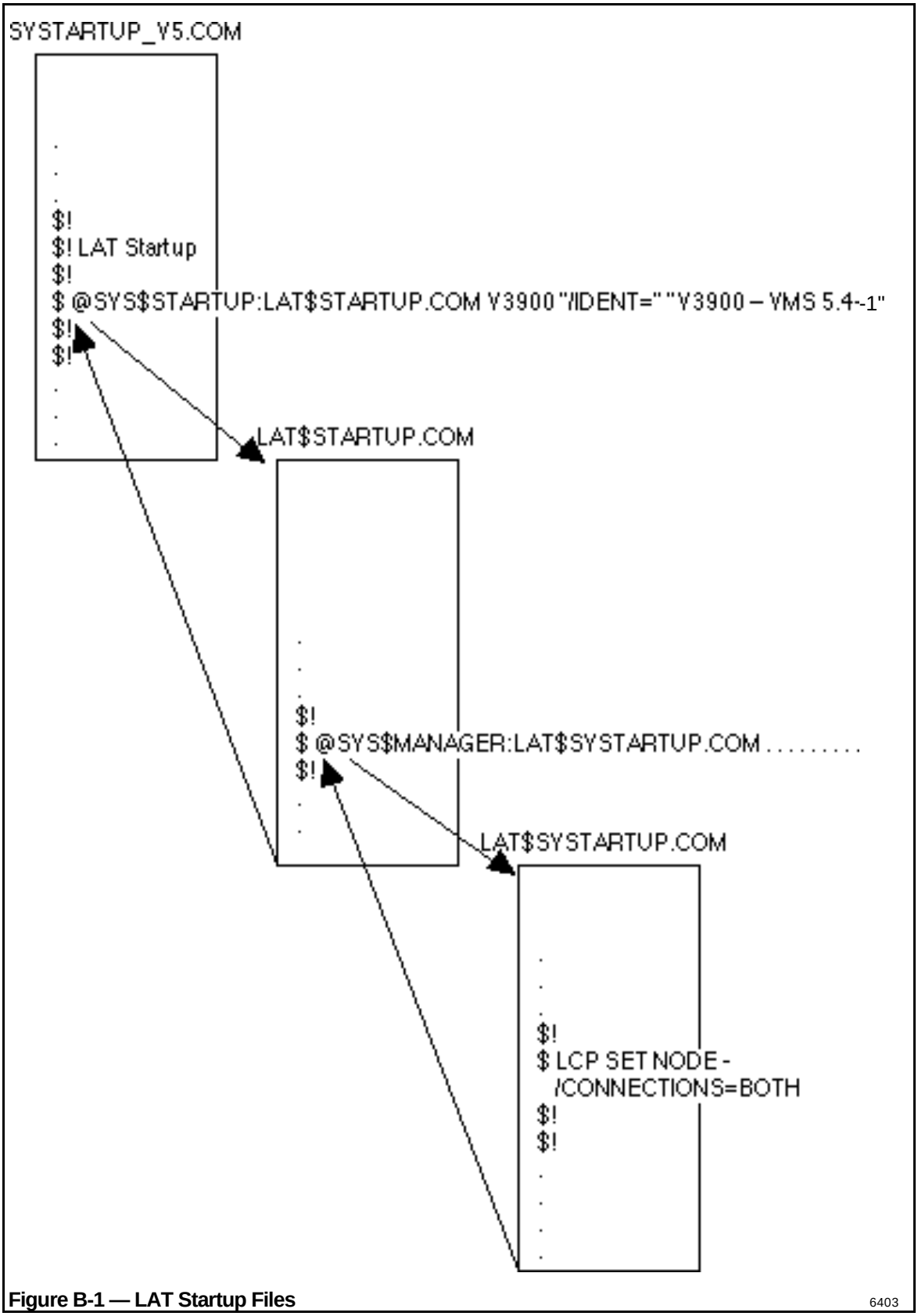


Figure B-1 — LAT Startup Files

`LCP> SET NODE/CONNECTIONS = BOTH`—tells this node that it can both initiate and accept connection requests. This is the proper CM50S/PLNM configuration. Executing the Show Services command will display all nodes on the LAT network that are offering their service.

This command must be added to the SYS$MANAGER:LAT$SYSTARTUP.COM file so that this is the configuration when the VAX is rebooted. If CM50S is started at or near the time that this command is executed (i.e., at system reboot), the initial attempt to connect to the PLNM service could fail since the service may not yet be visible to the host node. The PLNM broadcasts its service every minute, and when the host receives this broadcast it has visibility to the PLNM. The CM50S software retries the connection every 15 seconds and thus should connect to the PLNM within 1.25 minutes.

`LCP> SHOW PORT`—displays the current ports assigned on this node. There will be a port assigned to the PLNM node when CM50S is running. Using the `/full` option will display characteristics and counters that LATCP has compiled about the port.

`LCP> SET NODE/ENABLE = ()` or `LCP> SET NODE/DISABLE = ()`—allows you to selectively enable and disable LAT user groups and service groups. This allows you to optimize the load on the LTDRIVER process by filtering messages at the Ethernet controller. This is accomplished by configuring your node to receive only certain transmissions. By associating user group numbers and service group number to transmission, the Ethernet controller can ignore these transmissions at the lowest level instead of passing them to other processes to make that determination. This is an advanced feature used by experienced system/network managers.

The PLNM offers a service using transmission user group 0 and service group 0. The node from which CM50S is started must be configured to receive transmissions of this type. Again, this should be done in the SYS$MANAGER:LAT$SYSTARTUP.COM file to ensure that this is the default configuration after a reboot. If your node is not configured to receive transmissions on user group 0 and service group 0, then the PLNM service will be displayed by a Show Services command, yet the connection will fail during a CM50S startup.

`LCP> SHOW NODE`—displays the current node's configuration. This information shows the LAT Circuit Timer value, the connection configuration (i.e., incoming, outgoing), and the enabled and disabled user and service groups.

To exit LATCP, type either `EXIT` or `^Z`.

B.6.3.3 The CM50S Service

As mentioned previously, a VAX node broadcasts a service to allow other nodes to establish a connection. The PLNM broadcasts a service using the naming convention HVN_08002Bxxxxxx, where xxxxxx is replaced by the hardware address that is stenciled on the top of the rt-300. After the rt-300 has initialized itself, it broadcasts this service and is ready to accept a connection request.

If you execute a `$MCR LATCP SHOW SERVICES` command, the service is displayed. This service is visible by all nodes in the LAT network; the first CM50S startup requesting this service will have it granted. The second CM50S startup requesting a connection to this service will have it denied. If the CM50S startup fails to connect to the service, it will retry every 15 seconds until it is successful or CM50S is shut down.

For more than one PLNM link in the CM50S startup, one link can be successful while the other can fail. The first link will run normally while the second retries the connection. This retry adds only a slight overhead to the system.

If the link fails for any reason, such as a cable break or the PLNM is powered off or reset, then CM50S will output a disconnect message and start retrying to connect. Meanwhile, the PLNM displays a COMM DOWN state on the LCN operator station.

When connection is successful, CM50S outputs a LINK REINITIALIZATION message. When PLNM loading starts, CM50S outputs a RESTART IN PROGRESS message. When PLNM loading is comnplete, CM50S outputs a RESTART COMPLETE message, and the PLNM displays an OK state. At this time the communication link is established.

B.6.3.4 LAT Communications and Performance

There are several ways to tune your LAT network to achieve higher performance for the CM50S/PLNM connection, but the most impactive performance parameter is the LAT Circuit Timer.

The LAT Circuit Timer is the polling interval at which the Ethernet controller samples the LAT ports for activity. This changeable interval ranges between 10 and 1000 milliseconds. For a CM50S/PLNM connection, the lower the timer, the higher the throughput. **Note that the lowest timer value currently supported is 20 milliseconds.**

To modify the LAT Circuit Timer, you must enter LATCP and execute the following commands **from the console**.

```
LCP> STOP NODE
LCP> SET NODE/ CIRCUIT_TIMER = 20
LCP> START NODE
```

These commands stop the LTDRIVER process and modify the the LAT Circuit Timer (in this example, to a value of 20 milliseconds), then restarts the LTDRIVER process. Remember that the nominal interval range is between 10 and 1000 milliseconds.

It is necessary to stop LAT since the timer is not a dynamic parameter. This procedure must be done from the console since stopping the LTDRIVER process disconnects all LAT connections to the node. Therefore it is necessary that you notify any other users on the system to log off, or that you perform these commands during off-hours.

In addition, you will also need to add the following command to the SYS$MANAGER:LAT$SYSTARTUP.COM file (following the SET NODE/CONNECTIONS=BOTH command) to keep this configuration when the VAX is rebooted:

```
$ LCP SET NODE/CIRCUIT_TIMER = ##
```

It is not necessary to stop the LTDRIVER process since it is not running at this point in the system startup.

Lowering the Circuit Timer value has no adverse effects on performance by the rest of the system since the Ethernet controller is its own CPU. The Ethernet controller attempts to optimize transactions within the Circuit Timer interval by combining smaller transactions into larger transactions and sending only one packet. Evaluation of CM50S performance has shown that LAT handles larger transactions much more efficiently than smaller transactions.

B.7 DATA LINK STATUS INFORMATION

The CG holds data link status information in its Processor Status data point. To view this data, place the point $PRSTSnn (where nn is the CG node number) in a custom schematic display. The three data-link status parameters of this point and their value meanings are

ULP_STS = IN_SERV—Communications with the VAX have been established. This value is set when Restart is complete.
=FAILED—Communications with the VAX are broken. This value is set when a single link or both sides of a dual link have failed.

DL1_STS = IN_SERV—Link 1 has been connected and the CG is using or trying to use the link.
=FAILED—The CG has disconnected Link 1 because of problems.

DL2_STS = IN_SERV—Link 2 has been connected and the CG is using or trying to use the link.
=FAILED—The CG has disconnected Link 2 because of problems.
=NOT_INST—The CG has been configured for operation on Link 1 only (normal state for CG-VAX).

B.8 SYSTEM BACKUP

In order to enhance performance, CM50S keeps some data files open even when no applications are accessing them. Therefore, when backing up the system, always use the optional version of the Backup command: BACKUP/IGNORE=INTERLOCK.

CM50S CAPACITIES SUMMARY
Appendix C

This appendix summarizes the most important size and timing limits of the CG-VAX.

The values shown are permissible limits. The actual limits may be less depending on the combination of elements in a given database.

CG DATABASE LIMITS

ACIDPs per CG	250
CRDPs per CG	500
CG-resident DDTs	40
Units assigned to a CG	63

VAX/VMS DATABASE LIMITS

ACPs Installed	1000
Points per Input or Output DDT	300
Points per History DDT	24
DDTs Installed	2000

ACP SCHEDULING LIMITS

Delay maximum	60 seconds
Cyclic Program Run Intervals	10 seconds minimum, 24 hours maximum
Concurrent processes (active or hibernating) with connections to a CG/PLNM	47

MESSAGE SIZE LIMITS

For display	60 characters
For printing	72 characters
For archiving	120 characters

CUSTOM DATA SEGMENT LIMITS

CDS Parameter names	1000 per system
CDS per ACIDP or CRDP	10

CG MEMORY USE ESTIMATING

There are two sets of limits that must be considered in preparing the CG database: first, the design limits on the number of each item that is allowed; second, the amount of memory available and the size requirements for each item.

The built-in design limits (further restricted by memory availability) are

250 ACIDPs
500 CRDPs
10 Custom Data Segments per ACIDP or CRDP
40 CG-resident DDT/IDBs
300 parameters in a DDT/IDB
63 Units assigned to a CG

The amount of memory available in the CG for database use varies by product release. See the applicable LCN Release Guide for information about use of the CG size display to determine memory availability. Database memory requirements follow.

ACIDP—143 words each (plus CDS requirements)
CRDP—36 words each
DDT/IDB—40 + (25*the number of items in table) words each
Custom Data Segments—three factors are involved:

- Control Structure: 80 words of control structure are added to each ACIDP or CRDP with one or more CDS.

- Descriptor Segment: 13 +(22 times the number of parameters) words are required by each CDS for its descriptor segment.

- Data space: CDS data use varies with the mix of parameter types.
  Each Real parameter requires 2 words
  Each ASCII parameter requires 12 words
  Each Enumeration parameter requires 1 word
  Each Self-defining enumeration requires 4 words
  Each Boolean parameter requires 1 word

Process Units—300 words for each 150 points (or fraction of 150) in the same Unit.

CALLABLE FUNCTIONS AND PROCEDURES LIST
Appendix D

This appendix lists all the CM50S user-callable functions and procedures with cross references to detailed descriptions elsewhere in this manual.

Name	Description	Heading (FORTRAN / Pascal / C)
CM50_ABORT_TRANSFER	Abort LCN File Transfer	12.4.14/16.12.14/20.12.14
CM50_ACP_ACT	Activate (run) an ACP	12.1.3 / 16.1.3 / 20.1.3
CM50_ACP_CHG_MODE	Change ACP installation mode	12.1.7 / 16.1.7 / 20.1.7
CM50_ACP_CONNECT	Connect an ACP to an ACIDP	12.1.5/16.1.5/20.1.5
CM50_ACP_DEACTIVATE	Deactivate (abort) an ACP	12.1.4 / 16.1.4 / 20.1.4
CM50_ACP_DISCON	Disconnect ACP from its ACIDP	12.1.6 / 16.1.6 / 20.1.6
CM50_ACP_INSTALL	Install an ACP	12.1.1 / 16.1.1 / 20.1.1
CM50_ACP_LISTALL	Get list of ACPs	12.1.9 / 16.1.9 / 20.1.9
CM50_ACP_SUM	Get ACP summary	12.1.8 / 16.1.8 / 20.1.8
CM50_ACP_UNINST	Uninstall an ACP	12.1.2 / 16.1.2 / 20.1.2
CM50_ACPDELAY	ACP Delay	11.1.3 / 15.1.3 / 19.1.3
CM50_ATTR_LIST	List LCN File Attributes	12.4.3/16.4.3/20.4.3
CM50_CG_ACIDP	Get list of ACIDPs	12.3.4 / 16.3.4 / 20.3.4
CM50_CG_ADETAIL	Get detailed ACIDP information	12.3.3 / 16.3.3 / 20.3.3
CM50_CG_CONFIG	Get LCN configuration	12.3.5/16.3.5/20.3.5
CM50_CG_CRDP	Get list of CRDPs	12.3.2 / 16.3.2 / 20.3.2
CM50_CG_RDDT	Get list of resident DDTs	12.3.1 / 16.3.1 / 20.3.1
CM50_CONV_PT	Convert External to Internal ID	11.2.1 / 15.2.1 / 19.2.1
CM50_CONV_PT_LIST	Convert List of External IDs	11.2.2 / 15.2.2 / 19.2.2
CM50_CONV_TAG	Convert External to Internal ID	11.2.1 / 15.2.1 / 19.2.1
CM50_CONV_TAG_LIST	Convert List of External IDs	11.2.2 / 15.2.2 / 19.2.2
CM50_DATA_OUT	LCN Dataout Status	12.4.13/16.4.13/20.4.13
CM50_DDT_BUILD	Build / Rebuild a DDT	12.2.1 / 16.2.1 / 20.2.1
CM50_DDT_CONNECT	Connect a DDT to an ACIDP	12.2.6/16.2.6/20.2.6
CM50_DDT_DELETE	Delete a DDT	12.2.2 / 16.2.2 / 20.2.2
CM50_DDT_DETAIL	Get DDT detailed information	12.2.5 / 16.2.5 / 20.2.5
CM50_DDT_DISCONNECT	Disconnect a DDT from an ACIDP	12.2.7/16.2.7/20.2.7
CM50_DDT_GET	DDT Get Data	10.1.1 / 14.1.1 / 18.1.1
CM50_DDT_GETGEN	Generic DDT Get Data	10.1.3 / 14.1.3 / 18.1.3
CM50_DDT_GETNT	DDT Get Data	10.1.1 / 14.1.1 / 18.1.1
CM50_DDT_INSTALL	Install a DDT as CG resident	12.2.9 / 16.2.9 / 20.2.9
CM50_DDT_LIST	Get list of DDT summaries	12.2.4 / 16.2.4 / 20.2.4
CM50_DDT_STORE	DDT Store Data	10.1.2 / 14.1.2 / 18.1.2
CM50_DDT_STOREGEN	Generic DDT Store Data	10.1.4 / 14.1.4 / 18.1.4
CM50_DDT_STORENT	DDT Store Data	10.1.2 / 14.1.2 / 18.1.2
CM50_DDT_SUM	Get DDT summary information	12.2.3 / 16.2.3 / 20.2.3
CM50_DDT_TRIGGERS	Modify DDT prefetch Triggers	12.2.8/16.2.8/20.2.8
CM50_DDT_UNINST	Remove a DDT from CG residency	12.2.10 / 16.2.10 / 20.2.10
CM50_DDTHIS_AVER	Get History Averages (Rel. Time)	10.5.4 / 14.5.4 / 18.5.4
CM50_DDTHIS_AVERT	Get History Averages (Abs. Time)	10.5.5 / 14.5.5 / 18.5.5
CM50_DDTHIS_FAST	Get Fast History Snapshots (Rel.)	10.5.2/14.5.2/18.5.2
CM50_DDTHIS_FASTT	Get Fast History Snapshots (Abs.)	10.5.3/14.5.3/18.5.3
CM50_DDTHIS_MNTH	Get Monthly Averages (Rel. Time)	10.5.6/14.5.6/18.5.6
CM50_DDTHIS_MNTHT	Get Monthly Averages (Abs. Time)	10.5.7/14.5.7/18.5.7
CM50_DDTHIS_RATE	Find History Collection Rate	10.5.8/14.5.8/18.5.8

CM50_DDTHIS_SNAP	Get History Snapshots (Rel. Time)	10.5.2 / 14.5.2 / 18.5.2
CM50_DDTHIS_SNAPT	Get History Snapshots (Abs. Time)	10.5.3 / 14.5.3 / 18.5.3
CM50_FILE_CATALOG	List LCN Files to Dataout	12.4.6/16.4.6/20.4.6
CM50_FILE_LIST	List LCN Files & Extensions	12.4.4/16.4.4/20.4.4
CM50_GET_ASC24	Point List Get Data—ASCII	10.2.2/14.2.2/18.2.2
CM50_GET_ENUM	Point List Get Data—Enumeration	10.2.2/14.2.2/18.2.2
CM50_GET_EXID	Point List Get External IDs	10.2.2/14.2.2/18.2.2
CM50_GET_ID	Single Point Get Data(External ID)	10.3.1/14.3.1/18.3.1
CM50_GET_INTNBR	Point List Get Data—Integer	10.2.2/14.2.2/18.2.2
CM50_GET_ORD	Point List Get Data—Ordinal	10.2.2/14.2.2/18.2.2
CM50_GET_PT_LIST	Point List Get Data	10.2.1/14.2.1/18.2.1
CM50_GET_PTID	Point List Get Internal IDs	10.2.2/14.2.2/18.2.2
CM50_GET_REALNBR	Point List Get Data—Real	10.2.2/14.2.2/18.2.2
CM50_GET_STRI	Point List Get String Values	10.2.2/14.2.2/18.2.2
CM50_GET_TAG	Single Point Get Data(External ID)	10.3.1 / 14.3.1 / 18.3.1
CM50_GET_TIME	Point List Get Time Values	10.2.2/14.2.2/18.2.2
CM50_GETMSG	Get Message	10.6.1 / 14.6.1 / 18.6.1
CM50_GETPT_ID	Single Point Get Data (Internal ID)	10.3.3 / 14.3.3 / 18.3.3
CM50_HIBER	ACP Hibernate	11.1.4 / 15.1.4 / 19.1.4
CM50_HM_LIST	List LCN Volumes/Directories	12.4.5/16.4.5/20.4.5
CM50_LCN_COPY	LCN File Copy	12.4.8/16.4.8/20.4.8
CM50_LCN_DELETE	LCN File Delete	12.4.11/16.4.11/20.4.11
CM50_LCN_DIRECTORY	LCN Directory Maintenance	12.4.12/16.4.12/20.4.12
CM50_LCN_MOVE	LCN File Move	12.4.9/16.4.9/20.4.9
CM50_LCN_READ	Read File from LCN	12.4.1/16.4.1/20.4.1
CM50_LCN_RENAME	LCN File Rename	12.4.10/16.4.10/20.4.10
CM50_LCN_WRITE	Write File to LCN	12.4.2/16.4.2/20.4.2
CM50_MPL_GENFILE	Generate Multi-Point List—Text File	10.1.7 / 14.1.7 / 18.1.7
CM50_MPL_GENINCL	Create Include File for Multi-Point List	10.1.10 / 14.1.10 / 18.1.10
CM50_MPL_GENLIST	Generate Multi-Point List—ID Blocks	10.1.7 / 14.1.7 / 18.1.7
CM50_MPL_GENTAGS	Generate Multi-Point List—Tag Names	10.1.7 / 14.1.7 / 18.1.7
CM50_MPL_GET	Multi-Point List Get Data	10.1.5 / 14.1.5 / 18.1.5
CM50_MPL_READ	Read Multi-Point List	10.1.8 / 14.1.8 / 18.1.8
CM50_MPL_STORE	Multi-Point List Store Data	10.1.6 / 14.1.6 / 18.1.6
CM50_MPL_WRITE	Write Multi-Point List	10.1.9 / 14.1.9 / 18.1.9
CM50_MPLHIS_AVER	Get History Averages (Rel. Time)	10.5.4 / 14.5.4 / 18.5.4
CM50_MPLHIS_AVERT	Get History Averages (Abs. Time)	10.5.5 / 14.5.5 / 18.5.5
CM50_MPLHIS_MNTH	Get Monthly Averages (Rel. Time)	10.5.6/14.5.6/18.5.6
CM50_MPLHIS_MNTHT	Get Monthly Averages (Abs. Time)	10.5.7/14.5.7/18.5.7
CM50_MPLHIS_RATE	Find History Collection Rate	10.5.8/14.5.8/18.5.8
CM50_MPLHIS_SNAP	Get History Snapshots (Rel. Time)	10.5.2 / 14.5.2 / 18.5.2
CM50_MPLHIS_SNAPT	Get History Snapshots (Abs. Time)	10.5.3 / 14.5.3 / 18.5.3
CM50_PTHIS_AVER	Get History Averages (Rel. Time)	10.5.4 / 14.5.4 / 18.5.4
CM50_PTHIS_AVERT	Get History Averages (Abs. Time)	10.5.5 / 14.5.5 / 18.5.5
CM50_PTHIS_MNTH	Get Monthly Averages (Rel. Time)	10.5.6/14.5.6/18.5.6
CM50_PTHIS_MNTHT	Get Monthly Averages (Abs. Time)	10.5.7/14.5.7/18.5.7
CM50_PTHIS_RATE	Find History Collection Rate	10.5.8/14.5.8/18.5.8
CM50_PTHIS_SNAP	Get History Snapshots (Rel. Time)	10.5.2 / 14.5.2 / 18.5.2
CM50_PTHIS_SNAPT	Get History Snapshots (Abs. Time)	10.5.3 / 14.5.3 / 18.5.3
CM50_SET_ACP	ACP Initialization	11.1.1 / 15.1.1 / 19.1.1
CM50_SETBAD	Set Bad Value	11.3.2 / 15.3.2 / 19.3.2
CM50_SPCRAW	Convert Raw Data	10.4.3 / 14.4.3 / 18.4.3
CM50_SPGRAW	Raw Data Get	10.4.1 / 14.4.1 / 18.4.1
CM50_SPSRAW	Raw Data Store	10.4.2 / 14.4.2 / 18.4.2
CM50_STORE_ASC24	Point List Store Data—ASCII	10.2.4/14.2.4/18.2.4
CM50_STORE_ENUM	Point List Store Data—Enumeration	10.2.4/14.2.4/18.2.4
CM50_STORE_ID	Single Point Store Data(External ID)	10.3.2 / 14.3.2 / 18.3.2
CM50_STORE_INTNBR	Point List Store Data—Integer	10.2.4/14.2.4/18.2.4
CM50_STORE_ORD	Point List Store Data—Ordinal	10.2.4/14.2.4/18.2.4

CM50_STORE_PT_LIST	Point List Get Data	10.2.3/14.2.3/18.2.3
CM50_STORE_PTID	Point List Store Internal IDs	10.2.4/14.2.4/18.2.4
CM50_STORE_REALNBR	Point List Store Data—Real	10.2.4/14.2.4/18.2.4
CM50_STORE_STRI	Point List Store String Values	10.2.4/14.2.4/18.2.4
CM50_STORE_TAG	Single Point Store Data(External ID)	10.3.2 / 14.3.2 / 18.3.2
CM50_STORE_TIME	Point List Store Time Values	10.2.4/14.2.4/18.2.4
CM50_STOREMSG	Send Message	10.6.2 / 14.6.2 / 18.6.2
CM50_STOREPT_ID	Single Point Store Data (Internal ID)	10.3.4 / 14.3.4 / 18.3.4
CM50_TAGHIS_RATE	Find History Collection Rate	10.5.8/14.5.8/18.5.8
CM50_TIMARY_ASC	Conv Integer arr Time to ASCII	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMARY_EURO	Conv Integer arr Time to Euro str	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMARY_LCN	Conv Integer arr Time to LCN int	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMARY_VAXA	Conv Integer arr Time to VAX disp	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMARY_VAXB	Conv Integer arr Time to VAX bin	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMASC_ARY	Conv ASCII Time to Integer arr	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMASC_EURO	Conv ASCII Time to Euro str	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMASC_LCN	Conv ASCII Time to LCN int	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMASC_VAXA	Conv ASCII Time to VAX disp	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMASC_VAXB	Conv ASCII Time to VAX bin	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMEURO_ARY	Conv Euro Time to Integer arr	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMEURO_ASC	Conv Euro Time to ASCII	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMEURO_LCN	Conv Euro Time to LCN int	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMEURO_VAXA	Conv Euro Time to VAX disp	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMEURO_VAXB	Conv Euro Time to VAX bin	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMLCN_ARY	Conv LCN Time to Integer arr	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMLCN_ASC	Conv LCN Time to ASCII	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMLCN_EURO	Conv LCN Time to Euro str	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMLCN_VAXA	Conv LCN Time to VAX disp	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMLCN_VAXB	Conv LCN Time to VAX bin	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMNOW_ASC	Get LCN Clock Value (ASCII format)	10.3.5 / 14.3.5 / 18.3.5
CM50_TIMNOW_LCN	Get LCN Clock Value (internal format)	10.3.5 / 14.3.5 / 18.3.5
CM50_TIMVAXA_ARY	Conv VAX disp Time to Integer arr	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMVAXA_ASC	Conv VAX disp Time to ASCII	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMVAXA_EURO	Conv VAX disp Time to Euro str	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMVAXA_LCN	Conv VAX disp Time to LCN int	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMVAXA_VAXB	Conv VAX disp Time to VAX bin	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMVAXB_ARY	Conv VAX bin Time to Integer arr	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMVAXB_ASC	Conv VAX bin Time to ASCII	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMVAXB_EURO	Conv VAX bin Time to Euro str	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMVAXB_LCN	Conv VAX bin Time to LCN int	11.3.3 / 15.3.3 / 19.3.3
CM50_TIMVAXB_VAXA	Conv VAX bin Time to VAX disp	11.3.3 / 15.3.3 / 19.3.3
CM50_VALIDN	Valid Number Check	11.3.1 / 15.3.1 / 19.3.1
CM50_VOLUME_CATALOG	List LCN Volumes to Dataout	12.4.7/16.4.7/20.4.7
GETSTS	Get ACP Status	11.1.2 / 15.1.2 / 19.1.2
PRGTRM	ACP Termination	11.1.5 / 15.1.5 / 19.1.5

ASSIGNMENT OF PROCESS UNITS TO CG
Appendix E

This appendix contains information relating to the assignment of process units to the CG.

1. The number of process units that can be assigned to a CG is 63 minus the number of CGs assigned to the same checkpoint volume. For example, if you have two CGs assigned to one HM for checkpointing, there can be no more than 61 units assigned.

2. Because the CG node status display shows only points that are assigned to its area and to the CG node, you can never see more than 36 (area limit).

3. ACP access to parameters in other LCN nodes, both read and write, is independent of which units are assigned to the CG. That is, it does not matter to which unit the ACP's ACIDP is assigned.

4. Event-Initiated Processing from the HG and CL Messages from the AM and MC are independent of process unit assignment in the CG.

5. The operator demand of an ACP from its ACIDP detail display requires that the ACIDP is in a unit assigned to that Universal Station's area, or that the US is in Engineer keylock level. The same limitation applies to any CG point.parameter that you may wish to store to from a custom display.

6. An ACP can send an operator message to only the Unit its ACIDP is assigned to. Thus, only areas with that unit assigned will receive the message.

7. A process unit can be assigned to only one CG. If there are multiple CGs on the LCN, each must have a unique assignment of process units.

To summarize, for most flexibility, all units assigned to CGs should be assigned to all areas that need to communicate with that CG. The units that contain the points that are accessed by the CG do not have to be assigned to that CG.

In most cases, there is no real need to assign more than one unit to a CG.

INSTALLATION OF HDLC MODULES
Appendix F

The contents of this Appendix explain the installation of the HDLC Modules for three types of VAX data bus:

- *The VAX BI Bus*
- *The VAX Q-Bus*
- *The VAX UNIBUS*

F.1 INSTALLATION OF HDLC MODULE ON VAX BI BUS

The HDLC module functionally provides an interface between a VAX BI bus and TDC 3000 Computer Gateways. This document provides sufficient detail to install an HDLC module in standard VAXBI configurations. It assumes the installer is already familiar with installing options in VAXBI based processors.

Installation of the HDLC module requires three steps:

- Verification of resources
- Inserting the board
- Installing the interface connector.

All of the communication parameters such as clock source, speed, and interface are controlled by software. There are no switches to be set on the board.

F.1.1 Verification of Resources

The first step is to verify that the computer has sufficient resources for the installation. These resources include power, board slots, and I/O panel space.

1.	Power Requirements	+5 Vdc, 6 amps typical, 10 amps Max. -12 Vdc, 0.5 amps typical, 1.0 amp Max.
2.	Bus Slot	Single BI Slot
3.	I/O Panel	3 IOCP panel spaces

Verify that sufficient resources and board slots exist for your particular processor. Any question about the capacity of your existing CPU should be referred to the System Engineer.

BE VERY CAREFUL in computing the power capacity of the CPU buses. Do not exceed any of the limitations listed in the *Hardware Information Manual* for your particular processor. In particular, DO NOT USE the VAX Systems and Options Catalog because the power capacity of CPUs and buses are subject to change.

Next, verify that your processor has sufficient mounting space for the relevant hardware.

CAUTION

Static discharges can damage your board! Take all standard antistatic precautions. Use of the DEC-supplied ground straps and pads is mandatory. Wear the ESD wrist strap! Whenever you remove a circuit board from a VAXBI card cage, place it in a conductive container.

F.1.2 Inserting the HDLC Board

1. After verifying the correct operation of the system, completely power off the system.
2. Set the system circuit breaker in the rear to OFF.
3. Open the VAXBI chassis and locate an open slot.
4. Make note of the NODE ID plug; this information is required for software installation.
5. Lift the locking lever to open the slot.
6. Slide the module into the card cage slot until it stops. Note that this is a zero insertion force card cage.
7. Close the locking lever.
8. Restore the BI covers.

F.1.3 Installing the Interface Connector

1. Remove three IOCP Panels from the VAXBI enclosure.
2. Route the interface cables from the BI slot to the opening of the IOCP panel.
3. Connect the cables at the VAXBI Transition Header as labeled.
4. Connect the appropriate cables to the termination panel as labeled.
5. Mount the panel to the IOCP.
6. Restore power to the CPU.
7. The HDLC module is tested as part of the CM50S software installation.

You may now connect the Computer Gateway(s) to their ports (see heading F.4)

F.2 INSTALLATION OF HDLC MODULE ON Q-BUS

The HDLC module functionally provides an interface between a Q-bus and TDC 3000 Computer Gateways. This document provides sufficient detail to install an HDLC module in standard configurations. It assumes the installer is already familiar with installing options in VAX based processors.

Installation of the HDLC module requires three steps:

- Verification of resources
- Inserting the board
- Installing the interface connector.

Some of the communication parameters such as clock source and interface are controlled by switches.

F.2.1 Verification of Resources

The first step is to verify that the computer has sufficient resources for the installation. These resources include power, board slots, and I/O panel space.

1.	Power Requirements	+5 Vdc 5.5 amps Max. +12 Vdc 0.2 amps Max.
2.	Bus Slot	Quad Module
3.	I/O Panel	One panel space

Verify that sufficient resources and board slots exist for your particular processor. Be sure to account for +12 Volt power. Any question about the capacity of your existing CPU should be referred to the System Engineer.

BE VERY CAREFUL in computing the power capacity of the CPU buses. Do not exceed any of the limitations listed in the *Hardware Information Manual* for your particular processor or BA11. In particular, DO NOT USE the VAX Systems and Options Catalog because the power capacity of CPUs and buses are subject to change.

Next, verify that your processor has sufficient mounting space for the relevant hardware.

CAUTION

Static discharges can damage your board! Take all standard antistatic precautions. Use of the DEC-supplied ground straps and pads is mandatory. Wear the ESD wrist strap! Whenever you remove a circuit board from a VAX card cage, place it in a conductive container.

The HDLC module must reside at one of the addresses specified in the following list:

BOARD SLOT	DEVICE	CSR (OCTAL)	VECTOR (OCTAL)
0	ZQA0	761000	740-340
1	ZQB0	761002	744-344
2	ZQC0	761004	750-350
3	ZQD0	761006	754-354
4	ZQE0	761010	760-360
5	ZQF0	761012	764-364
6	ZQG0	761014	770-370
7	ZQH0	761016	774-374

As supplied, the board's interrupt vector is 7xx. The vector can be changed to 3xx by cutting the jumper located next to pins 14 and 15 of U010. If you have conflicts in device addresses from nonstandard peripherals use the next available location from table 3. To determine the configuration of a system issue the following command from a login with cmkrnl privilege:

```
$mc sysgen
SYSGEN>  SHOW /CONFIG
```

The VAX responds with the CSRs and Vectors for all currently installed devices, for example:

```
System CSR and Vectors on  2-MAR-1988 14:50:11.22
Name:PUA Units: 1 Nexus:0 (UBA) CSR: 772150 Vector1: 774 Vector2: 000
Name:PTA Units: 1 Nexus:0 (UBA) CSR: 774500 Vector1: 260 Vector2: 000
Name:XQA Units: 2 Nexus:0 (UBA) CSR: 774440 Vector1: 120 Vector2: 000
Name:TTA Units: 4 Nexus:0 (UBA) CSR: 760100 Vector1: 300 Vector2: 304
Name:PUB Units: 1 Nexus:0 (UBA) CSR: 760354 Vector1: 310 Vector2: 000
SYSGEN>
```

Verify that your selected CSR from the table does not conflict with existing units. Set the CSR module dip switch SWA located near the board edge connector as specified in the following table.

CSR Module	SWA Switch Position		
	1	2	3
ZQA0	OPEN	OPEN	OPEN
ZQB0	OPEN	OPEN	CLOSED
ZQC0	OPEN	CLOSED	OPEN
ZQD0	OPEN	CLOSED	CLOSED
ZQE0	CLOSED	OPEN	OPEN
ZQF0	CLOSED	OPEN	CLOSED
ZQG0	CLOSED	CLOSED	OPEN
ZQH0	CLOSED	CLOSED	CLOSED

Use A Pen or screwdriver to change DIP switch settings. DO NOT USE A PENCIL. The remaining switches SW0 through SW7 should be set to 1-closed 2-open (that is, Position 1 is down toward the LED side of the board, Position 2 is down on the side furthest away from the LED side of the board).

F.2.2 Inserting the HDLC Board

1. After verifying the correct operation of the system, completely power off the system.
2. Set the system circuit breaker in the rear to OFF.
3. Open the Q-BUS chassis and locate an open slot.
4. Remove the bus grant card from the open slot.
5. Remove the NPG jumper from CA1 to CB1 on the rear of the card cage.
6. Insert the module in the slot, being careful not to contact the adjacent modules.

F.2.3 Installing the Interface Connector

1. Remove three IOCP Panels from the VAX enclosure.
2. Route the interface cables from the Q-BUS slot to the opening of the IOCP panel.
3. Connect the cables at the Q-BUS as labeled.
4. Connect the appropriate cables to the termination panel as labeled.
5. Mount the panel to the IOCP.
6. Restore Q-BUS covers.
7. Restore power to the CPU.
8. The HDLC module is tested as part of the CM50S software installation.

You may now connect the Computer Gateway(s) to their ports (see heading F.4)

F.3 INSTALLATION OF HDLC MODULE ON VAX UNIBUS

The HDLC module functionally provides an interface between a VAX UNIBUS bus and TDC 3000 Computer Gateways. This document provides sufficient detail to install an HDLC module in standard configurations. It assumes the installer is already familiar with installing options in VAX based processors.

Installation of the HDLC module requires three steps:

- Verification of resources
- Inserting the board
- Installing the interface connector.

Some of the communication parameters such as clock source and interface are controlled by switches.

F.3.1 Verification of Resources

The first step is one of verifying that the computer has sufficient resources for the installation. These resources include power, board slots, and I/O panel space.

1.	Power Requirements	+5 Vdc, 5.5 amps Max. -15 Vdc, 0.2 amps Max.
2.	Bus Slot	Single HEX SPC slot
3.	I/O Panel	3 IOCP panel spaces

Verify that sufficient resources and board slots exist for your particular processor. Be sure to account for -15 Volt power. Any question about the capacity of your existing CPU should be referred to the System Engineer.

BE VERY CAREFUL in computing the power capacity of the CPU busses. Do not exceed any of the limitations listed in the *Hardware Information Manual* for your particular processor or BA11. In particular, DO NOT USE the VAX Systems and Options Catalog because the power capacity of CPUs and buses are subject to change.

Next, verify that your processor has sufficient mounting space for the relevant hardware.

CAUTION

Static discharges can damage your board! Take all standard antistatic precautions. Use of the DEC-supplied ground straps and pads is mandatory. Wear the ESD wrist strap! Whenever you remove a circuit board from a VAX card cage, place it in a conductive container.

The HDLC module must reside at one of the addresses specified in the the following list:

1. 762000-762176 ZPA0
2. 762200-762376 ZPB0
3. 762400-762576 ZPC0
4. 762600-762776 ZPD0
5. 763000-763176 ZPE0
6. 763200-762376 ZPF0
7. 763400-763576 ZPG0
8. 763600-762776 ZPH0

If you have conflicts in device addresses from nonstandard peripherals use the next available location. To determine the configuration of a system, issue the following command from a login with cmkrnl privilege:

```
$ mc sysgen
SYSGEN>  SHOW /CONFIG
```

The VAX responds with the CSR and Vectors for all currently installed devices, for example:

```
System CSR and Vectors on  2-MAR-1988 14:50:11.22
Name:PUA Units: 1 Nexus:0 (UBA) CSR: 772150 Vector1: 774 Vector2: 000
Name:PTA Units: 1 Nexus:0 (UBA) CSR: 774500 Vector1: 260 Vector2: 000
Name:XQA Units: 2 Nexus:0 (UBA) CSR: 774440 Vector1: 120 Vector2: 000
Name:TTA Units: 4 Nexus:0 (UBA) CSR: 760100 Vector1: 300 Vector2: 304
Name:PUB Units: 1 Nexus:0 (UBA) CSR: 760354 Vector1: 310 Vector2: 000
SYSGEN>
```

Verify that your selected CSR from the table does not conflict with existing units. Set the CSR module dip switch SWA located near the board edge connector as follows:

CSR Module	SWA Switch Position		
	1	2	3
ZPA0	OPEN	OPEN	OPEN
ZPB0	OPEN	OPEN	CLOSED
ZPC0	OPEN	CLOSED	OPEN
ZPD0	OPEN	CLOSED	CLOSED
ZPE0	CLOSED	OPEN	OPEN
ZPF0	CLOSED	OPEN	CLOSED
ZPG0	CLOSED	CLOSED	OPEN
ZPH0	CLOSED	CLOSED	CLOSED

Use a pen or screwdriver to change DIP switch settings. DO NOT USE A PENCIL. The remaining switches SW0 through SW7 should be set to 1-closed 2-open (that is, Position 1 is down toward the LED side of the board, Position 2 is down on the side furthest away from the LED side of the board).

F.3.2 Inserting the HDLC Board

1. After verifying the correct operation of the system, completely power off the system.
2. Set the system circuit breaker in the rear to OFF.
3. Open the UNIBUS chassis and locate an open slot.
4. Remove the bus grant card from the open slot.
5. Remove the NPG jumper from CA1 to CB1 on the rear of the card cage.
6. Insert the module in the slot, being careful not to contact the adjacent modules.

F.3.3 Installing the Interface Connector

1. Remove three IOCP Panels from the VAX enclosure.
2. Route the interface cables from the UNIBUS slot to the opening of the IOCP panel.
3. Connect the cables at the UNIBUS as labeled.
4. Connect the appropriate cables to the termination panel as labeled.
5. Mount the panel to the IOCP.
6. Restore Unibus covers.
7. Restore power to the CPU.
8. The HDLC module is tested as part of the CM50S software installation.

You may now connect the Computer Gateway(s) to their ports (see heading F.4)

F.4 HDLC CONNECTIONS TO COMPUTER GATEWAYS

Within CM50S, each CG is addressed by a logical CG Port number (1 to 4). Any CG Port number can be assigned to any physical link on any installed HDLC board; however, the CG Port numbers must be assigned sequentially.

Even if no physical connection is made for CG Port number 1, it will have to be assigned before CG Port number 2. That is, the system will not address CG Port #2 unless a CG Port #1 has been assigned.

Use the CM50S HDLC INSTALLATION WORKSHEET (on the following page) to keep track of the assignment of your CG Ports and HDLC links.

Each CG should be connected (by a direct cable connection or a fiber optics modem) to one of the physical links on the termination panel for an HDLC board. For BI Bus and Q-Bus boards, the links are numbered 0 to 3. For a UNIBUS board the links are numbered 0 to 7.

The choice of physical link connections has an impact on the maximum communications baud rate. On a BI Bus or Q-Bus board, only links 2 and 3 may be used for speeds above 19200 baud. On a UNIBUS board, only links 6 and 7 may operate at speeds above 19200 baud. Note that if more than two CG Ports are connected to one HDLC board, then none of those CG boards will operate faster than 19200 baud.

CM50S HDLC INSTALLATION WORKSHEET

Hardware installation information:

HDLC BOARD 1: Device Name = _____

BI Bus Node = _____(decimal)

or Qbus/UNIBUS CSR = _____(octal)

vector = _____(octal)

CG connections:

CG Port #1: Board Device Name = _____

Physical Link = _____

Direct or Modem Connect: _____

Baud Rate = _____

CG Port #2: Board Device Name = _____

Physical Link = _____

Direct or Modem Connect: _____

Baud Rate = _____

CG Port #3: Board Device Name = _____

Physical Link = _____

Direct or Modem Connect: _____

Baud Rate = _____

CG Port #4: Board Device Name = _____

Physical Link = _____

Direct or Modem Connect: _____

Baud Rate = _____

VINTAGE PROCEDURES
Appendix G

The information in this appendix provides information on all CM50S Release 1 calls and those Release 2 interface calls that have been replaced in this release.

G.1 INTRODUCTION TO VINTAGE INTERFACE ROUTINES

This appendix contains information on the use of all user interface routines available at Rel 1 of CM50S and those Rel 2 interface routines that have been replaced by improved functions at Rel 3. All of these routines continue to be supported by Rel 3 as a convenience for early system users.

NOTE

The Rel 1 user-program interface-routines are all implemented as external FORTRAN subroutines; therefore, any Pascal program (ACP) that uses any of these interfaces **must** contain one of the following include statements in its declaration section:

```
%INCLUDE 'CM50$LIB:CM50_INCLUDE.PAS'
```

or `%INCLUDE 'CM50$LIB:PASCAL.INCLUDE'`

G.1.1 Summary of Rel 1 User-Program Interfaces

Paragraph	Interface Descriptions	Interface Names FORTRAN	Interface Names Pascal
G.2	Multipoint Data Transfers		
G.2.1	Get Data	GETDTF/GETDTF1	GETDTA/GETDTA1
G.2.2	Store Data	STRDTF/STRDTF1	STRDTA/STRDTA1
G.3	Single Point Data Transfers		
G.3.1	Get Single Point (External ID)	SPGETF	SP_GET
G.3.2	Store Single Point (External ID)	SPSTRF	SP_STR
G.3.3	Get Single Point (Internal ID)	SPIGTF	SP_IGT
G.3.4	Store Single Point (Internal ID)	SPISTF	SP_IST
G.4	History Data Transfers		
G.4.1	Get History (Absolute Times)	HISABF	HISABP
G.4.2	Get History (Relative Time)	HISRLF	HISRLP
G.5	Text Message Transfers		
G.5.1	Get Message	GETMSF	GETMSG
G.5.2	Send Message	SNDMSF	SNDMSG
G.6	ACP Execution Support		
G.6.1	ACP Trap Handler	ACPTRP	ACPTRP
G.6.2	Get ACP Status	GETSTS	GETSTS
G.6.3	ACP Delay	PRGDLY	PRGDLY
G.6.4	ACP Termination	PRGTRM	PRGTRM
G.7	Utility Routines		
G.7.1	Check Bad Value	CHKBAD	CHKBAD
G.7.2	Set Bad Value	SETBAD	SETBAD
G.7.3	Convert External to Internal ID	SPCVTF	SP_CVT
G.7.4	Convert LCN Time	CONVTF	CONVTP

G.1.2 Summary of Replaced Rel 2 User-Program Interfaces

Paragraph	Interface Descriptions	Interface Names
G.8	Single Point Data Transfers	
G.8.1	Get Single Point (External ID)	CM50_GETPT
G.8.2	Store Single Point (External ID)	CM50_STOREPT
G.8.3	Get LCN Clock Value	CM50_GETPT
G.9	Point Arrays without DDTs	
G.9.1	Point List Get Data	CM50_GET_REAL CM50_GET_INTG CM50_GET_ASCI CM50_GET_AENM CM50_GET_OENM
G.9.2	Point List Store Data	CM50_STORE_REAL CM50_STORE_INTG CM50_STORE_ASCI CM50_STORE_AENM CM50_STORE_OENM
G.10	Single-Point Get History	
G.10.1	Single-Point Get History	CM50_GET_HIS CM50_GET_REAL_HIS CM50_GET_INTG_HIS
G.11	ACP Execution Support	
G.11.1	ACP Trap Handler	ACPTRP
G.12	Utility Routines	
G.12.1	Convert External to Internal ID	CM50_CONV_ID CM50_CONV_TP

G.1.3 Summary of Replaced Rel 3 User-Program Interfaces

Paragraph	Interface Descriptions	Interface Names
G.13.1	Connect ACP to ACIDP	CM50_ACP_CON
G.13.2	Connect DDT to ACIDP	CM50_DDT_CON
G.13.3	Disconnect DDT from ACIDP	CM50_DDT_DISCON
G.13.4	Modify Triggers	CM50_MOD_TRIGGERS

G.2 MULTIPOINT DATA TRANSFERS

The interface routines in this group require the use of separately prepared Data Definition Tables (DDT) that specify which points are to be accessed and what pre/post processing is to be done on data values.

Single elements of parameter arrays (but not whole arrays) can be specified in the DDT.

G.2.1 Get Data Interface

This routine fetches data from the DDTs associated CG or elsewhere on its LCN. The specification of which data is to be fetched and where it is to be stored in the calling program's data arrays is contained in the Data Definition Table referenced by the call.

> **NOTE**
>
> Use the Interface Name GETDTF or GETDTA if you want data transformation operations performed by the Table Processor, and GETDTF1 or GETDTA1 if you do not want transformation operations performed.

G.2.1.1 Pascal Procedure Call for Get Data

```
GETDTA (DATA_TABLE_NAME, REAL_VALUES_ARRAY, INTEGER_VALUES_ARRAY,
        %DESCR ASCII_VALUES_ARRAY, %REF ENUM_VALUES_ARRAY or
        %REF ENUM_ORD_ARRAY, PRIORITY, STATUS_TABLE, RETURN_STATUS);
```

The call must contain the equivalent of either ENUM_VALUES_ARRAY or ENUM_ORD_ARRAY, but not both, and must match the DDT when it includes either ENUMER or ORDINAL points. Note the required use of %DESCR or %REF with certain parameters.

G.2.1.2 Typical Pascal Data Definitions for Get Data

```
TYPE
  INTEGER_2_TYPE      : [WORD] -32768..32767;

VAR
  DATA_TABLE_NAME     : PACKED ARRAY [1..9] OF CHAR;
  REAL_VALUES_ARRAY   : ARRAY [1..300] OF REAL;
  INTEGER_VALUES_ARRAY: ARRAY [1..300] OF INTEGER_2_TYPE;
  ASCII_VALUES_ARRAY  : ARRAY [1..300] OF ASCII_VALS;
  ENUM_VALUES_ARRAY   : ARRAY [1..300] OF ENUM_VALS;      } (One only, must
  ENUM_ORD_ARRAY      : ARRAY [1..300] OF INTEGER_2_TYPE; } match DDT)
  PRIORITY            : INTEGER_2_TYPE;
  STATUS_TABLE        : ARRAY [1..300] OF INTEGER_2_TYPE;
  RETURN_STATUS       : INTEGER_2_TYPE;
  ASCII_VALS          : PACKED ARRAY [1..24] OF CHAR;
  ENUM_VALS           : PACKED ARRAY [1..8]  OF CHAR;
```

The type-specification values for INTEGER_2_TYPE, ASCII_VALS, ENUM_VALS and DATA_TABLE_NAME are fixed. The others show maximum values.

G.2.1.3 FORTRAN Subroutine Call for Get Data

```
CALL GETDTF   (DATA_TABLE_NAME, REAL_VALUES_ARRAY, INTEGER_VALUES_ARRAY,
              ASCII_VALUES_ARRAY, %REF(ENUM_VALUES_ARRAY) or
              %REF(ENUM_ORD_ARRAY), PRIORITY, STATUS_TABLE,
              RETURN_STATUS)
```

You must include either %REF(ENUM_VALUES_ARRAY) or %REF(ENUM_ORD_ARRAY), but not both, and must match the DDT if it includes either ENUMER or ORDINAL points.

G.2.1.4 Typical FORTRAN Data Definitions for Get Data

```
CHARACTER*9       DATA_TABLE_NAME
REAL*4            REAL_VALUES_ARRAY(300)
INTEGER*2         INTEGER_VALUES_ARRAY(300)
CHARACTER*24      ASCII_VALUES_ARRAY(300)
CHARACTER*8       ENUM_VALUES_ARRAY(300)   } (One only, must
INTEGER*2         ENUM_ORD_ARRAY(300)      } match DDT)
INTEGER*2         PRIORITY
INTEGER*2         STATUS_TABLE(300)
INTEGER*2         RETURN_STATUS
```

G.2.1.5 Parameter Definitions for Get Data

DATA_TABLE_NAME—The name of a packed array of ASCII characters that contains the name of the input Data Definition Table to be used. The table name is a maximum of nine characters long.

REAL_VALUES_ARRAY—The name of a single-dimension array into which the fetched Real values are to be stored. Bad values are returned as NaN (-0).

INTEGER_VALUES_ARRAY—The name of a single-dimension Integer array into which the fetched 16-bit Integer values are to be stored.

ASCII_VALUES_ARRAY—The name of a packed array into which fetched ASCII strings are to be stored. Bad values are returned as strings of 24 question marks.

ENUM_VALUES_ARRAY—The name of a packed array into which fetched Enumeration strings are to be stored. Bad values are returned as strings of eight question marks.

ENUM_ORD_ARRAY—The name of a single-dimension array of 16-bit Integers into which the fetched ordinal values of the enumerations are to be stored.

PRIORITY—The name of a 16-bit Integer variable that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

STATUS_TABLE—The name of a single-dimension array of 16-bit Integers for the storage of returned point-related error/status information. A status code is returned for each requested value.

RETURN_STATUS—The name of a 16-bit Integer variable where the request completion status is to be stored. The assigned meanings are

0 – Normal return—request successfully completed.
1 – Referenced DDT does not exist or is incomplete.
3 – Unable to access the LCN, or data link failed, or mailbox does not exist.
4 – Referenced DDT is not of type "input," or ACIDP not in RUN state.
5 – Request completed, but errors exist in data.
6 – Referenced CG-resident DDT was not found.
7 – ACP not installed or is restricted or was not turned on by CM50S function.
8 – Referenced DDT is "in use."
9 – Illegal priority number (must be 1 or 2).
10 – Node not present
16 – Specified ACIDP not found in CG.
28 – Invalid data access priority
33 – Invalid data type specified in DDT/IDB
35 – DDT/IDB purpose not same as request
36 – DDT/IDB header number of words disagrees with table content
37 – Data type repeated in DDT/IDB
38 – DDT/IDB contains both enumeration and ordinal data
39 – DDT/IDB contains over 300 points
40 – Number of words specified in message header disagrees with message content.

G.2.2 Store Data Interface

This routine sends data to points in the DDTs associated CG or elsewhere on its LCN. The specification of what points are to receive data and the location of data within the calling program's data arrays is contained in the Data Definition Table referenced by the call. Errors encountered during execution of the routine as well as individual point-data errors are returned to the calling program.

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode.

> **NOTE**
>
> Use the Interface Name STRDTF or STRDTA if you want data transformation operations performed by the Table Processor, and STRDTF1 or STRDTA1 if you do not want transformation operations performed.

G.2.2.1 Pascal Procedure Call for Store Data

```
STRDTA (DATA_TABLE_NAME, REAL_VALUES_ARRAY, INTEGER_VALUES_ARRAY,
        %DESCR ASCII_VALUES_ARRAY, %REF ENUM_VALUES_ARRAY or
        %REF ENUM_ORD_ARRAY, PRIORITY, STORE_CODE_TABLE,
        STATUS_TABLE, RETURN_STATUS);
```

The call must contain the equivalent of either ENUM_VALUES_ARRAY or ENUM_ORD_ARRAY, but not both, and must match the DDT if it includes either ENUMER or ORDINAL points. Note the required use of %DESCR or %REF with certain parameters.

G.2.2.2 Typical Pascal Data Definitions for Store Data

```
TYPE
  INTEGER_2_TYPE      : [WORD] -32768..32767;

VAR
  DATA_TABLE_NAME     : PACKED ARRAY [1..9]   OF CHAR;
  REAL_VALUES_ARRAY   : ARRAY [1..300] OF REAL;
  INTEGER_VALUES_ARRAY: ARRAY [1..300] OF INTEGER_2_TYPE;
  ASCII_VALUES_ARRAY  : ARRAY [1..300] OF ASCII_VALS;
  ENUM_VALUES_ARRAY   : ARRAY [1..300] OF ENUM_VALS;      } (One only, must
  ENUM_ORD_ARRAY      : ARRAY [1..300] OF INTEGER_2_TYPE; } match DDT)
  PRIORITY            : INTEGER_2_TYPE;
  STORE_CODE_TABLE    : ARRAY [1..300] OF INTEGER_2_TYPE;
  STATUS_TABLE        : ARRAY [1..300] OF INTEGER_2_TYPE;
  RETURN_STATUS       : INTEGER_2_TYPE);
  ASCII_VALS          : PACKED ARRAY [1..24]  OF CHAR;
  ENUM_VALS           : PACKED ARRAY [1..8]   OF CHAR;
```

The type-specification values for INTEGER_2_TYPE, ASCII_VALS, ENUM_VALS and DATA_TABLE_NAME are fixed; the others show maximum values.

G.2.2.3 FORTRAN Subroutine Call for Store Data

```
CALL STRDTF (DATA_TABLE_NAME, REAL_VALUES_ARRAY, INTEGER_VALUES_ARRAY,
             ASCII_VALUES_ARRAY, %REF(ENUM_VALUES_ARRAY) or
             %REF(ENUM_ORD_ARRAY),PRIORITY, STORE_CODE_TABLE,
             STATUS_TABLE, RETURN_STATUS)
```

You must include either %REF(ENUM_VALUES_ARRAY) or %REF(ENUM_ORD_ARRAY), but not both, and must match the DDT if it includes either ENUMER or ORDINAL points.

G.2.2.4 Typical FORTRAN Data Definitions for Store Data

```
CHARACTER*9       DATA_TABLE_NAME
REAL*4            REAL_VALUES_ARRAY(300)
INTEGER*2         INTEGER_VALUES_ARRAY(300)
CHARACTER*24      ASCII_VALUES_ARRAY(300)
CHARACTER*8       ENUM_VALUES_ARRAY(300)  } (One only, must
INTEGER*2         ENUM_ORD_ARRAY(300)     } match DDT)
INTEGER*2         PRIORITY
INTEGER*2         STORE_CODE_TABLE(300)
INTEGER*2         STATUS_TABLE(300)
INTEGER*2         RETURN_STATUS
```

G.2.2.5 Parameter Definitions for Store Data

DATA_TABLE_NAME—The name of a packed array of ASCII characters that contains the name of the output Data Definition Table to be used in the "Store Data" operation. The table name is a maximum of nine characters long.

REAL_VALUES_ARRAY—The name of a single-dimension array that contains the Real values to be stored.

INTEGER_VALUES_ARRAY—The name of a single-dimension array of 16-bit Integers that contains the Integer values to be stored.

ASCII_VALUES_ARRAY—The name of a packed array that contains the ASCII strings to be stored.

ENUM_VALUES_ARRAY—The name of a packed array that contains the Enumeration strings to be stored. Use of enumeration strings by Store Data is limited to standard enumerations (including Custom Data Segments). All self-defined enumerations (such as digitals) must be accessed through their ordinal values.

ENUM_ORD_ARRAY—The name of a single-dimension array of 16-bit integers that contains the Ordinal values of the Enumerations that are to be stored.

PRIORITY—The name of a 16-bit Integer variable that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

STORE_CODE_TABLE—The name of a single-dimension array of 16-bit Integers that contains a control code entry for each value to be stored. These codes control what—if any—value is to be stored. The store code values are

0 – Store the value from the Values Array
1 – Store the bad value representation instead
2 – Do not store any value.

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

STATUS_TABLE—The name of a single-dimension array of 16-bit integers for the storage of returned point-related error/status information. A status code is returned for each requested store value.

RETURN_STATUS—The name of a 16-bit Integer variable where the request completion status is to be stored. The assigned meanings are

0 – Normal return—request was successfully completed.
1 – Referenced Data Definition Table does not exist or is incomplete.
3 – Unable to access the LCN, or data link failed or mailbox does not exist.
4 – Referenced Data Definition Table is not of type "output."
5 – Request completed, but errors exist in data.
6 – Specified ACIDP not found, or ACP not connected to an ACIDP.
7 – Program is not installed, or is in "restricted" mode, or was not turned on by CM50S function, or ACP execution state in the CG is not correct.
8 – Referenced Data Definition Table is "in use" or ACIDP's access key is set to read only.
9 – Illegal priority number (must be 1 or 2).
15 – ACP connected ACIDP not in RUN state.
25 – Data must be enumeration type
28 – Invalid data access priority
33 – Invalid data type specified in DDT/IDB
35 – DDT/IDB purpose not same as request
36 – DDT/IDB header number of words disagrees with table content
37 – Data type repeated in DDT/IDB
38 – DDT/IDB contains both enumeration and ordinal data
39 – DDT/IDB contains over 300 points
40 – Number of words specified in message header disagrees with message content

G.3 SINGLE POINT DATA TRANSFERS

The interface routines in this group Get or Store values from or to, one named point.parameter (or parameter array) at a time. For parameter arrays, up to the whole array is accessed. The External ID version of Get Single Point is also used to get LCN date and time.

G.3.1 Get Single Point (External ID) Interface

This routine fetches data for a single point from a specified CG or elsewhere on its LCN. The specification of which data is to be fetched and where it is to be stored is contained in the call. For parameter arrays, either a single element, the whole array, or an array subset starting with the first element can be specified.

G.3.1.1 Pascal Procedure Call for Get Single Point

```
SP_GET (POINT_ID, POINT_ENT_INDEX, %REF PT_PARAM, PARAM_INDEX,
        %REF VALUE_LOC, VALUE_TYPE, PRIORITY, VALUE_STATUS, IUNIT,
        RETURN_STATUS);
```

Note the required use of %REF with certain parameters.

G.3.1.2 Typical Pascal Data Definitions for Get Single Point

```
TYPE
   INTEGER_2_TYPE  : [WORD] -32678..32767;

VAR
   POINT_ID        : PACKED ARRAY [1..8]  OF CHAR;
   POINT_ENT_INDEX : INTEGER_2_TYPE;
   PT_PARAM        : PACKED ARRAY [1..8]  OF CHAR;
   PARAM_INDEX     : INTEGER_2_TYPE;
   VALUE_LOC       : (See Parameter Definitions);
   VALUE_TYPE      : INTEGER_2_TYPE;
   PRIORITY        : INTEGER_2_TYPE;
   VALUE_STATUS    : INTEGER_2_TYPE;
   IUNIT           : INTEGER_2_TYPE;
   RETURN_STATUS   : INTEGER_2_TYPE;
```

The data definitions that follow are examples for VALUE_LOC.

```
   ASCII_VALS      : PACKED ARRAY [1..24] OF CHAR;
   ENUM_VALS       : PACKED ARRAY [1..8]  OF CHAR;
   TIME_ARR        : PACKED ARRAY [1..17] OF CHAR;
   REAL_ARR        : ARRAY [1..n] OF REAL;
   INTEGER_ARR     : ARRAY [1..n] OF INTEGER_2_TYPE;
   ENUM_ARR        : ARRAY [1..n] OF ENUM_VALS;
   ORD_ARR         : ARRAY [1..n] OF INTEGER_2_TYPE;
```

G.3.1.3 FORTRAN Subroutine Call for Get Single Point

```
CALL SPGETF   (POINT_ID, POINT_ENT_INDEX, PT_PARAM, PARAM_INDEX,
               VALUE_LOC, VALUE_TYPE, PRIORITY, VALUE_STATUS,
               IUNIT, RETURN_STATUS)
```

Note that if VALUE_TYPE equals 3, 4, 6, or 9 (character string types), the FORTRAN call parameter must use pass by reference for VALUE_LOC, i.e., `%REF(VALUE_LOC)`.

G.3.1.4 Typical FORTRAN Data Definitions for Get Single Point

```
CHARACTER*8         POINT_ID
INTEGER*2           POINT_ENT_INDEX
CHARACTER*8         PT_PARAM
INTEGER*2           PARAM_INDEX
(See Param Definitions) VALUE_LOC
INTEGER*2           VALUE_TYPE
INTEGER*2           PRIORITY
INTEGER*2           VALUE_STATUS
INTEGER*2           IUNIT
INTEGER*2           RETURN_STATUS
```

G.3.1.5 Parameter Definitions for Get Single Point

POINT_ID—The name of a packed array of eight characters that contains the ASCII Point ID (name) of the point from which the value is to be retrieved.

POINT_ENT_INDEX—The name of a 16-bit Integer variable that contains an index for use with point arrays. This feature is not implemented and the value must be zero.

PT_PARAM—The name of a packed array of eight characters that contains the ASCII name of a parameter (or parameter array) from which the value(s) is retrieved.

PARAM_INDEX—The name of a 16-bit Integer variable whose interpretation is controlled by VALUE_TYPE.

When VALUE_TYPE is 1, 2, 3, 4, or 5, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of PARAM_INDEX is used as an index and must be greater than zero. If the parameter being accessed is not an array type, PARAM_INDEX must be zero.

When VALUE_TYPE is 7, 8, 9, or 10, a whole array (or a subset of the array starting with the first element) is to be accessed and PARAM_INDEX is used to specify the number of elements to be accessed—If PARAM_INDEX is smaller than the actual array size, only that number of elements is returned. If it is larger than the actual array size, RETURN_STATUS equals 14 (no elements are returned).

VALUE_LOC—The name of a program variable into which the value(s) are to be stored. The type of variable must match what is declared in VALUE_TYPE.

VALUE_TYPE	VALUE_LOC type
1	Real (32 bits)
2	Integer (16 bits)
3	ASCII_VALS (24 characters)
4	ENUM_VALS (8 characters)
5	Integer (16 bits)
6	TIME_ARR (18 characters, see note following)
7	ARRAY [1..n] OF REAL
8	ARRAY [1..n] OF INTEGER
9	ARRAY [1..n] OF ENUM_VALS
10	ARRAY [1..n] OF INTEGER

VALUE_TYPE—The name of a 16-bit Integer variable that contains a number that designates value type of the accessed parameters as follows:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
6 = LCN time (see following note)
7 = Real array
8 = Integer array
9 = Enumeration array
10 = Ordinal value of enumeration array

NOTE

For LCN Time requests, only the following parameter contents are significant: VALUE_LOC (6), VALUE_TYPE (6), PRIORITY (1), VALUE_STATUS, and RETURN_STATUS. The other parameters, including POINT_ID, should be blank. The format of TIME_ARR values is MM/DD/YY∆HH:MM:SS∆ (where ∆ is used to indicate a space).

PRIORITY—The name of an Integer variable that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

VALUE_STATUS—The name of a 16-bit Integer variable into which point-related status information is to be stored. This value is meaningful only when the RETURN_STATUS value is 0 (normal), 5 (complete with errors), or 14 (array size error). For any other RETURN_STATUS, VALUE_STATUS is zero (0). When the VALUE_TYPE is an array, VALUE_STATUS refers to status of the whole array.

IUNIT—The name of a 16-bit Integer variable (1-4) identifying the CG to be accessed.

RETURN_STATUS—The name of a 16-bit Integer variable where the request-completion status is to be stored. The assigned meanings are

0 – Normal return—request successfully completed.
3 – Unable to access the LCN, or data link failed, or mailbox does not exist.
4 – ACP connected ACIDP not in RUN state.
5 – Data-link transaction complete, request had errors, see VALUE_STATUS.
6 – Point entity or parameter was not found on the LCN, or ACP connected ACIDP not found.
9 – Illegal priority number (must be 1 or 2).
10 – Invalid call parameters or array size is ≤ 0 or >1000.
11 – Invalid data type or data type does not match parameter type.
12 – Error when attempting to get a memory buffer in the VAX.
13 – CG message-transaction error—internal error.
14 – Array size error—more array elements were requested than exist in the array.
15 – Point.parameter ID or ASCII/Enumeration value not on word boundary.
22 – Data type out of range
28 – Invalid data access priority
29 – Invalid data type code
30 – Point name index must be zero
32 – Parameter array pointer/size too large
40 – Number of words specified in message header disagrees with message content

G.3.2 Store Single Point (External ID) Interface

This routine stores data to a single point in a specified CG or elsewhere on its LCN. The specification of where the data is to be found and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

G.3.2.1 Pascal Procedure Call for Store Single Point

```
SP_STR (POINT_ID, POINT_ENT_INDEX, PT_PARAM,
        PARAM_INDEX, %REF VALUE_LOC, VALUE_TYPE, PRIORITY,
        STORE_CODE, STORE_STATUS, RETURN_STATUS);
```

Note the required use of %REF with certain parameters.

G.3.2.2 Typical Pascal Data Definitions for Store Single Point

```
TYPE
   INTEGER_2_TYPE  : [WORD] -32768..32767;

VAR
   POINT_ID        : PACKED ARRAY [1..8]  OF CHAR;
   POINT_ENT_INDEX : INTEGER_2_TYPE;
   PT_PARAM        : PACKED ARRAY [1..8]  OF CHAR;
   PARAM_INDEX     : INTEGER_2_TYPE;
   VALUE_LOC       : (See Parameter Definitions);
   VALUE_TYPE      : INTEGER_2_TYPE;
   PRIORITY        : INTEGER_2_TYPE;
   STORE_CODE      : INTEGER_2_TYPE;
   STORE_STATUS    : INTEGER_2_TYPE;
   RETURN_STATUS   : INTEGER_2_TYPE;
```

The data definitions that follow are examples for VALUE_LOC.

```
   ASCII_VALS      : PACKED ARRAY [1..24] OF CHAR;
   ENUM_VALS       : PACKED ARRAY [1..8]  OF CHAR;
   REAL_ARR        : ARRAY [1..n] OF REAL;
   INTEGER_ARR     : ARRAY [1..n] OF INTEGER_2_TYPE;
   ENUM_ARR        : ARRAY [1..n] OF ENUM_VALS;
   ORD_ARR         : ARRAY [1..n] OF INTEGER_2_TYPE;
```

G.3.2.3 FORTRAN Subroutine Call for Store Single Point

```
CALL SPSTRF   (POINT_ID, POINT_ENT_INDEX, PT_PARAM, PARAM_INDEX,
               VALUE_LOC, VALUE_TYPE, PRIORITY, STORE_CODE,
               STORE_STATUS, RETURN_STATUS)
```

Note that if VALUE_TYPE equals 3, 4, or 9 (character string types), the FORTRAN call parameter must use pass by reference for VALUE_LOC, i.e., `%REF(VALUE_LOC)`.

G.3.2.4 Typical FORTRAN Data Definitions for Store Single Point

```
CHARACTER*8         POINT_ID
INTEGER*2           POINT_ENT_INDEX
CHARACTER*8         PT_PARAM
INTEGER*2           PARAM_INDEX
(See Param Definitions) VALUE_LOC
INTEGER*2           VALUE_TYPE
INTEGER*2           PRIORITY
INTEGER*2           STORE_CODE
INTEGER*2           STORE_STATUS
INTEGER*2           RETURN_STATUS
```

G.3.2.5 Parameter Definitions for Store Single Point

POINT_ID—The name of a packed array of eight characters that contains the ASCII Point ID (name) of the point where the value is to be stored.

POINT_ENT_INDEX—The name of a 16-bit Integer variable that contains an index for use with point arrays. This feature is not implemented and the value must be zero.

PT_PARAM—The name of a packed array of eight characters that contains the ASCII parameter name for the point.parameter where the value is to be stored.

PARAM_INDEX—The name of a 16-bit Integer variable whose interpretation is controlled by VALUE_TYPE.

When VALUE_TYPE is 1, 2, 3, 4, or 5, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of PARAM_INDEX is used as an index and must be greater than zero. If the parameter being accessed is not an array type, the PARAM_INDEX value must be zero.

When VALUE_TYPE is 7, 8, 9, or 10, a whole array is to be accessed and PARAM_INDEX is used to specify the number of array elements—If PARAM_INDEX does not match the actual array size, no elements are stored and RETURN_STATUS value is 5 with a STORE_STATUS indicating an invalid array size.

VALUE_LOC—The name of a program variable from which the value(s) are to be fetched. The type of variable must match what is declared in VALUE_TYPE.

VALUE_TYPE	VALUE_LOC type
1	Real (32 bits)
2	Integer (16 bits)
3	ASCII_VALS (24 characters)
4	ENUM_VALS (8 characters)
5	Integer (16 bits)
6	not used
7	ARRAY [1..n] OF REAL
8	ARRAY [1..n] OF INTEGER
9	ARRAY [1..n] OF ENUM_VALS
10	ARRAY [1..n] OF INTEGER

VALUE_TYPE—The name of a 16-bit Integer variable that contains a number that designates value type as follows:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
6 = not used
7 = Real array (conversion at CG)
8 = Integer array
9 = Enumeration array
10 = Ordinal value of enumeration array

PRIORITY—The name of a 16-bit Integer variable that contains the requested data-access priority:

1 = High priority (provided for control operations)
2 = Low priority (provided for noncontrol operations)

STORE_CODE—Name of a 16-bit Integer variable that contains a code that allows the substitution of a bad value representation in place of the provided value(s). The store code values are

0 = Store the data value(s) provided
1 = Store the bad value representation instead

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

STORE_STATUS—The name of a 16-bit Integer variable into which point-related status information is to be stored. This value is meaningful only when the RETURN_STATUS value is 0 (normal) or 5 (complete with errors). For any other RETURN_STATUS, VALUE_STATUS is zero (0). See Appendix A for a listing of Data-Access error/status codes. When the VALUE_TYPE is an array, STORE_STATUS refers to status of the whole array.

RETURN_STATUS—The name of a 16-bit Integer variable where the request-completion status is to be stored. The assigned meanings are

0 – Normal return—request successfully completed.
3 – Unable to access the LCN, or data link failed, or mailbox does not exist.
4 – ACP connected ACIDP not in RUN state.
5 – Data-link transaction complete, request had errors, see STORE_ STATUS.
6 – Valid ACIDP not found, or point entity or parameter not found.
7 – ACP not installed, or is restricted, or was not turned on by CM50S function.
8 – ACIDP access key incorrect
9 – Illegal priority number (must be 1 or 2)
10 – Invalid call parameters, or store code is invalid, or array size is ≤0 or > 1000.
11 – Invalid data type, or data type does not match parameter type.
12 – Insufficient memory space in the VAX
13 – CG message transaction error—internal error
14 – Array size too small
15 – Point.parameter ID or ASCII/Enumeration value not on word boundary.
22 – Data type out of range
28 – Invalid data access priority
29 – Invalid data type code
30 – Point name index must be zero
32 – Parameter array pointer/size too large
40 – Number of words specified in message header disagrees with message content.

See also, error codes listed in paragraphs A.2, and A.3.

G.3.3 Get Single Point (Internal ID) Interface

This routine fetches data for a single point from the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the SPCVTF interface, see G.7.3) reduces the overhead required for repetitive single-point requests.

The specification of which data is to be fetched, and where it is to be stored, is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

G.3.3.1 Pascal Procedure Call for Get Single Point

```
SP_IGT (ID_BLOCK, %REF VALUE_LOC, PRIORITY, VALUE_STATUS, IUNIT,
        RETURN_STATUS);
```

Note the required use of %REF with certain parameters.

G.3.3.2 Typical Pascal Data Definitions for Get Single Point

```
TYPE
   INTEGER_2_TYPE  : [WORD] -32768..32767;

VAR
   ID_BLOCK        : ARRAY [1..8] OF INTEGER_2_TYPE;
   VALUE_LOC       : (See Parameter Definitions);
   PRIORITY        : INTEGER_2_TYPE;
   VALUE_STATUS    : INTEGER_2_TYPE;
   IUNIT           : INTEGER_2_TYPE;
   RETURN_STATUS   : INTEGER_2_TYPE;
```

The data definitions that follow are examples for VALUE_LOC.

```
   ASCII_VALS      : PACKED ARRAY [1..24] OF CHAR;
   ENUM_VALS       : PACKED ARRAY [1..8]  OF CHAR;
   REAL_ARR        : ARRAY [1..n] OF REAL;
   INTEGER_ARR     : ARRAY [1..n] OF INTEGER_2_TYPE;
   ENUM_ARR        : ARRAY [1..n] OF ENUM_VALS;
   ORD_ARR         : ARRAY [1..n] OF INTEGER_2_TYPE;
```

G.3.3.3 FORTRAN Subroutine Call for Get Single Point

```
CALL SPIGTF   (ID_BLOCK, VALUE_LOC, PRIORITY, VALUE_STATUS,
               IUNIT, RETURN_STATUS)
```

Note that if VALUE_TYPE equals 3, 4, or 9 (character string types), the FORTRAN call parameter must use pass by reference for VALUE_LOC, i.e., `%REF(VALUE_LOC)`.

G.3.3.4 Typical FORTRAN Data Definitions for Get Single Point

```
INTEGER*2          ID_BLOCK(8)
(See Param Definitions)  VALUE_LOC
INTEGER*2          PRIORITY
INTEGER*2          VALUE_STATUS
INTEGER*2          IUNIT
INTEGER*2          RETURN_STATUS
```

G.3.3.5 Parameter Definitions for Get Single Point

ID_BLOCK—The name of an array of eight 16-bit integer values that contains the internal ID data block obtained by a previous SPCVTF call. When the data is of array type, that call returns the array size in word 7 of the ID block. Thus, if you wish to get less than the entire array, you can change the parameter qualifier in the seventh word of the ID block to be smaller than the actual array size. Do not change any other words in the ID block. See paragraph 4.7.8 of the *Computer Gateway User Manual* for ID block details.

VALUE_LOC—The name of a program variable into which the value is to be stored. The type of variable must match that which was declared in VALUE_TYPE in the earlier SPCVTF call.

VALUE_TYPE	VALUE_LOC type
1	Real
2	Integer
3	ASCII_VALS
4	ENUM_VALS
5	Integer
6	not used
7	ARRAY [1..n] OF REAL
8	ARRAY [1..n] OF INTEGER
9	ARRAY [1..n] OF ENUM_VALS
10	ARRAY [1..n] OF INTEGER

PRIORITY—The name of a 16-bit Integer variable that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

VALUE_STATUS—The name of a 16-bit Integer variable into which point-related status information is to be stored. This value is meaningful only when the RETURN_STATUS value is 0 (normal), 5 (complete with errors), or 14 (array size error). For any other RETURN_STATUS value, VALUE_STATUS is zero. See Appendix A for a listing of Data-Access error/status codes. When the VALUE_TYPE is an array, VALUE_STATUS refers to status of the whole array.

IUNIT—The name of a 16-bit Integer variable (1-4) identifying the CG to be accessed.

RETURN_STATUS—The name of a 16-bit Integer variable where the request-completion status is to be stored. The assigned meanings are

0 – Normal return—request successfully completed
3 – Unable to access the LCN, or data link failed, or mailbox does not exist
4 – ACP connected ACIDP not in RUN state
5 – Data-link transaction complete, request had errors, see VALUE_STATUS.
6 – Point entity or parameter not found on the LCN, or ACP connected ACIDP not in RUN state.
9 – Illegal priority number (must be 1 or 2)
10 – Invalid parameters
11 – Invalid data type or data type does not match the parameter type.
12 – Error when attempting to get a memory buffer in the VAX
13 – Not the expected CG return transaction, internal CM50S communication error.
14 – Array size error—more array elements were requested than exist in the array. The returned VALUE_STATUS contains the actual array size.
15 – Point.parameter ID or ASCII/Enumeration value not on word boundary.
28 – Invalid data access priority
29 – Invalid data type code
40 – Number of words specified in message header disagrees with message content.

G.3.4 Store Single Point (Internal ID) Interface

This routine stores data to a single point in the CG or elsewhere on the LCN. Use of the Internal point.parameter ID (obtained by previous use of the SPCVTF interface, see G.7.3) reduces the overhead required for repetitive single-point requests.

The specification of where the data is found and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

G.3.4.1 Pascal Procedure Call for Store Single Point

```
SP_IST (ID_BLOCK, %REF VALUE_LOC, PRIORITY, STORE_CODE, STORE_STATUS,
        RETURN_STATUS);
```

G.3.4.2 Typical Pascal Data Definitions for Store Single Point

```
TYPE
   INTEGER_2_TYPE  : [WORD] -32768..32767;

VAR
   ID_BLOCK        : ARRAY [1..8] OF INTEGER_2_TYPE;
   VALUE_LOC       : (See Parameter Definitions);
   PRIORITY        : INTEGER_2_TYPE;
   STORE_CODE      : INTEGER_2_TYPE;
   STORE_STATUS    : INTEGER_2_TYPE;
   RETURN_STATUS   : INTEGER_2_TYPE;
```

The data definitions that follow are examples for VALUE_LOC.

```
   ASCII_VALS      : PACKED ARRAY [1..24] OF CHAR;
   ENUM_VALS       : PACKED ARRAY [1..8]  OF CHAR;
   REAL_ARR        : ARRAY [1..n] OF REAL;
   INTEGER_ARR     : ARRAY [1..n] OF INTEGER_2_TYPE;
   ENUM_ARR        : ARRAY [1..n] OF ENUM_VALS;
   ORD_ARR         : ARRAY [1..n] OF INTEGER_2_TYPE;
```

G.3.4.3 FORTRAN Subroutine Call for Store Single Point

```
CALL SPISTF   (ID BLOCK, VALUE_LOC, PRIORITY, STORE_CODE, STORE_STATUS,
              RETURN_STATUS)
```

Note that if VALUE_TYPE equals 3, 4, or 9 (character string types), the FORTRAN call parameter must use pass by reference for VALUE_LOC, i.e., `%REF(VALUE_LOC)`.

G.3.4.4 Typical FORTRAN Data Definitions for Store Single Point

```
INTEGER*2          ID_BLOCK(8)
(See Param Definitions)  VALUE_LOC
INTEGER*2          PRIORITY
INTEGER*2          STORE_CODE
INTEGER*2          STORE_STATUS
INTEGER*2          RETURN_STATUS
```

G.3.4.5 Parameter Definitions for Store Single Value

ID_BLOCK—The name of an array of eight, 16-bit integer values that contains the internal ID data block obtained by a previous SPCVTF call. Do not change any words in the ID block. If the array size is changed, the array is not stored and the RETURN_STATUS value is 5 with a STORE_STATUS that indicates an invalid array size. See paragraph 4.7.8 of the *Computer Gateway User Manual* for ID block details.

VALUE_LOC—The name of a program variable that contains the value to be stored. The type of variable must match what was declared in VALUE_TYPE in the earlier SPCVTF call.

VALUE_TYPE	VALUE_LOC type
1	Real
2	Integer
3	ASCII_VALS
4	ENUM_VALS
5	Integer
6	not used
7	ARRAY [1..n] OF REAL
8	ARRAY [1..n] OF INTEGER
9	ARRAY [1..n] OF ENUM_VALS
10	ARRAY [1..n] OF INTEGER

PRIORITY—The name of a 16-bit Integer variable that contains the requested data-access priority:

1 = High priority (provided for control operations)
2 = Low priority (provided for noncontrol operations)

STORE_CODE—The name of a 16-bit Integer variable that contains a code that allows the substitution of a bad value representation in place of the provided value(s). The store code values are

0 = Store the data value(s) provided
1 = Store the bad value representation instead

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

STORE_STATUS—The name of a 16-bit Integer variable into which point-related status information is to be stored. This value is meaningful only when the RETURN_STATUS value is 0 (normal), or 5 (complete with errors). For any other RETURN_STATUS, STORE_STATUS is zero. See Appendix A for a listing of Data-Access error/status codes. When the VALUE_TYPE is an array, STORE_STATUS refers to status of the whole array.

RETURN_STATUS—The name of a 16-bit Integer variable where the request-completion status is to be stored. The assigned meanings are

0 – Normal return—request successfully completed
3 – Unable to access the LCN, or data link failed, or mailbox does not exist
4 – ACP connected ACIDP not in RUN state
5 – Data-link transaction complete, request had errors, see STORE_STATUS.
6 – Valid ACIDP not found, or point entity or parameter not found on the LCN.
7 – ACP not installed, or not in normal status, or was not turned on by CM50S function
8 – ACIDP access key incorrect
9 – Illegal priority number (must be 1 or 2)
10 – Invalid call parameters
11 – Invalid data type or data type does not match parameter name.
12 – Error when attempting to acquire a memory buffer in the VAX
13 – CG message transaction error—internal error
15 – ASCII/Enumeration value not on word boundary
28 – Invalid data access priority
29 – Invalid data type code
40 – Number of words specified in message header disagrees with message content.

See also, error codes listed in paragraphs A.2, and A.3.

G.4 HISTORY DATA TRANSFERS

The interface routines in this group get previously stored averages or 1-minute snapshot data from the History Module. Use of separately prepared DDTs is required.

The calls described in paragraphs G.4.1 and G.4.2 provide for concurrent Get History requests by up-to-four ACPs. A fifth request is rejected with a queue-full status return. These calls differ only, in how the start and end times are specified. The form of returned data is the same for both.

G.4.1 Get History (Absolute Times) Interface

This routine is used to fetch history data from the HM using absolute begin and end times. The specification of which data is to be collected is contained in a Data Definition Table, while time bounds and the history type are calling parameters. There is a maximum of 262 samples that can be returned per call.

If a seasonal time change has occurred during a specified Absolute History interval, the number of samples returned can differ from the expected number of samples. For example, if it is desired to obtain a day's worth of hourly averages (24) and a forward time change of one hour has occurred, 23 samples are returned. If the time change is in the backward direction, 25 samples are returned.

G.4.1.1 Pascal Procedure call for Get History (Absolute Times)

```
HISABP (DATA_ARRAY_NAME, DATA_ARRAY_SIZE, DATA_TABLE_NAME,
        BEGIN_PERIOD, END_PERIOD, HISTORY_TYPE,
        NUMBER_OF_WORDS, RETURN_STATUS);
```

G.4.1.2 Typical Pascal Data Definitions for Get History (Absolute Times)

```
TYPE
   INTEGER_2_TYPE  : [WORD] -32768..32767;

VAR
   DATA_ARRAY_NAME : ARRAY [1..6000] OF INTEGER_2_TYPE;
   DATA_ARRAY_SIZE : INTEGER_2_TYPE;
   DATA_TABLE_NAME : PACKED ARRAY [1..9]  OF CHAR;
   BEGIN_PERIOD    : PACKED ARRAY [1..14] OF CHAR;
   END_PERIOD      : PACKED ARRAY [1..14] OF CHAR;
   HISTORY_TYPE    : INTEGER_2_TYPE;
   NUMBER_OF_WORDS : INTEGER_2_TYPE;
   RETURN_STATUS   : INTEGER_2_TYPE;
```

G.4.1.3 FORTRAN Subroutine Call for Get History (Absolute Times)

```
CALL HISABF   (DATA_ARRAY_NAME, DATA_ARRAY_SIZE, DATA_TABLE_NAME,
               BEGIN_PERIOD, END_PERIOD, HISTORY_TYPE, NUMBER_OF_WORDS,
               RETURN_STATUS)
```

G.4.1.4 Typical FORTRAN Data Definitions for Get History (Absolute Times)

```
INTEGER*2          DATA_ARRAY_NAME(6000)
INTEGER*2          DATA_ARRAY_SIZE
CHARACTER*9        DATA_TABLE_NAME
CHARACTER*14       BEGIN_PERIOD
CHARACTER*14       END_PERIOD
INTEGER*2          HISTORY_TYPE
INTEGER*2          NUMBER_OF_WORDS
INTEGER*2          RETURN_STATUS
```

G.4.1.5 Parameter Definitions for Get History (Absolute Times)

DATA_ARRAY_NAME—The name of a single-dimension Integer array where the history data is to be stored. A Get History call can return a maximum of 5986, 16-bit words of data.

DATA_ARRAY_SIZE—The name of a 16-bit Integer variable that contains the size in words of the data array.

DATA_TABLE_NAME—The name of a packed array of nine characters that contains the ASCII name of the DDT to be used.

BEGIN_PERIOD—The name of a packed array that contains the 14-character ASCII representation MM/DD/YYΔHH:MM (where Δ indicates a blank character), of the date and time starting the period for which history is to be fetched.

The value entered for this parameter should be some time after the previous period's collection, but before the desired time period begins. For example, to get an hourly average at 11:00, BEGIN_PERIOD should be set any time between 10:01 and 10:59.

END_PERIOD—The name of a packed array that contains the 14-character ASCII representation, MM/DD/YYΔHH:MM, of the date and time ending the period for which history is to be fetched.

The value entered for this parameter should be some time after the desired time period begins, but before the next time period's collection time. For example, END_PERIOD for an 11:00 hourly average should be set to any time from 11:01 to 11:59.

HISTORY_TYPE—The name of a 16-bit Integer variable that contains the number specifying the requested history type. The history types and their integer representations are

1-minute snapshots	= 0
Hourly averages	= 1
Shift averages	= 2
Daily averages	= 3
Monthly averages	= 4
User averages	= 5

NUMBER_OF_WORDS—The name of a 16-bit Integer variable where HISABP/HISABF is to store the total word count of history values returned to the program. Its maximum value is 5986. It is zero if the request completion status is other than 0 or 5.

RETURN_STATUS—The name of a 16-bit Integer variable where the request-completion status is to be stored. The assigned meanings are

0 – Normal return—request was successfully completed
1 – The named Data Definition Table does not exist or is incomplete.
2 – Referenced Data Definition Table is not of type "history."
3 – Unable to access the LCN, or Data Link failed, or mailbox does not exist.
4 – Begin Period is invalid.
5 – Request completed, but with errors.
6 – End Period is invalid.
7 – History Type is not valid or there was a Data Access failure during the history query.
9 – Could not obtain history data from the HM.
11 – Get History request timed out.
12 – Get History request queue is full.
13 – Begin/End periods are invalid.
14 – Array size is too small.
15 – ACP connected to ACIDP not in RUN state.
16 – Specified ACIDP not found

See also, error codes listed in paragraphs A.2, and A.3.

G.4.2 Get History (Relative Time) Interface

This routine is used to fetch history data from the HM, using a relative offset from current LCN time. The specification of which data is to be collected is contained in a Data Definition Table, while time bounds and the history type are calling parameters.

The direction of search can be either forward (oldest to newest data) or backwards (newest to oldest data); however, a forward search requires at least twice as long to execute. To execute a backward search, set the starting offset value less-than or equal-to the ending offset value.

The number of samples returned is calculated as the positive difference between the starting offset and the ending offset, plus one. If this difference exceeds 262, the request is truncated at 262 samples. The number of samples returned by a Relative History request is immune to time changes.

Offset values less than one have special meanings. When the starting or ending offset value is zero (i.e., current LCN time) in the case of averages, the first sample returned is the current running average for the period.

A starting offset of -1 has special meaning in the cases of snapshots and user averages. In those cases only, LCN time is rounded to the beginning of the last hour. This permits an ACP to be sure of obtaining the last full hour of snapshots or user averages. In calculating the number of samples returned, a -1 is treated as an offset of 0 and its number of samples and direction of search follow those rules.

An ending offset of -1 for snapshots and user averages means the search direction is forward and the ending time is on the hour starting "n" units back from current time.

The following table summarizes results of some possible combinations of starting and ending offsets with numbers of samples returned and reasons for zero sample returns.

History Type	Starting Offset	Ending Offset	Number of Samples	Direction of Search	Partial First Sample for Averages?
any	0	0	1	Backward	yes
any	1	1	1	Backward	no
any	2	3	2	Backward	no
any	3	2	2	Forward	no
any	0	300	262	Backward	yes
0,5	3	-1	4	Forward	no
1 to 4	3	-1	0	Error, end offset invalid	
0,5	-1	3	4	Backward	no
0,5	-1	-3	0	Error, end offset invalid	
1 to 4	-1	-3	0	Error, begin/end offset invalid	

G.4.2.1 Pascal Procedure Call for Get History (Relative Time)

```
HISRLP (DATA_ARRAY_NAME, DATA_ARRAY_SIZE, DATA_TABLE_NAME,
        BEGIN_OFFSET, END_OFFSET, HISTORY_TYPE, NUMBER_OF_WORDS,
        RETURN_STATUS);
```

G.4.2.2 Typical Pascal Data Definitions for Get History (Relative Time)

```
TYPE
   INTEGER_2_TYPE  : [WORD] -32768..32767;

VAR
   DATA_ARRAY_NAME : ARRAY [1..6000] OF INTEGER_2_TYPE;
   DATA_ARRAY_SIZE : INTEGER_2_TYPE;
   DATA_TABLE_NAME : PACKED ARRAY [1..9] OF CHAR;
   BEGIN_OFFSET    : INTEGER_2_TYPE;
   END_OFFSET      : INTEGER_2_TYPE;
   HISTORY_TYPE    : INTEGER_2_TYPE;
   NUMBER_OF_WORDS : INTEGER_2_TYPE;
   RETURN_STATUS   : INTEGER_2_TYPE;
```

G.4.2.3 FORTRAN Subroutine Call for Get History (Relative Time)

```
CALL HISRLF  (DATA_ARRAY_NAME, DATA_ARRAY_SIZE, DATA_TABLE_NAME,
              BEGIN_OFFSET, END_OFFSET, HISTORY_TYPE, NUMBER_OF_WORDS,
              RETURN_STATUS)
```

G.4.2.4 Typical FORTRAN Data Definitions for Get History (Relative Time)

```
INTEGER*2         DATA_ARRAY_NAME(6000)
INTEGER*2         DATA_ARRAY_SIZE
CHARACTER*9       DATA_TABLE_NAME
INTEGER*2         BEGIN_OFFSET
INTEGER*2         END_OFFSET
INTEGER*2         HISTORY_TYPE
INTEGER*2         NUMBER_OF_WORDS
INTEGER*2         RETURN_STATUS
```

G.4.2.5 Parameter Definitions for Get History (Relative Time)

DATA_ARRAY_NAME—The name of a single-dimension array of 16-bit integers where the history data is to be stored. A Get History call can return a maximum of 5986, 16-bit words of data.

DATA_ARRAY_SIZE—The name of a 16-bit Integer variable that contains the size in words of the data array.

DATA_TABLE_NAME—The name of a packed array of nine characters that contains the ASCII name of the DDT to be used in this Get History operation.

BEGIN_OFFSET—The name of a positive Integer variable that indicates a relative offset from current LCN time that represents the starting period for which history is to be fetched. The units for the offset are based on the type of history requested.

HISTORY TYPE	UNITS OF OFFSET
1-minute snapshots	Minutes
Hourly averages	Hours
Shift averages	not applicable
Daily averages	Days
Monthly averages	Months
User averages	Number of user averages to skip

END_OFFSET—The name of a positive Integer variable that indicates a relative offset from the current LCN time representing the ending period for which history is to be fetched. The units for the offset are based on the type of history requested. See values given with BEGIN_OFFSET.

HISTORY_TYPE—The name of a 16-bit Integer variable that contains the number specifying the requested history type. The history types and their integer representations are

1-minute snapshots	= 0
Hourly averages	= 1
Shift averages	= 2
Daily averages	= 3
Monthly averages	= 4
User averages	= 5

NUMBER_OF_WORDS—The name of a 16-bit Integer variable where HISRLP/HISRLF is to store the total word count of history values returned to the program. Its maximum value is 5986. It is zero if the request-completion status is other than 0 or 5.

RETURN_STATUS—The name of a 16-bit Integer variable where the request-completion status is to be stored. The assigned meanings are

0 – Normal return—request was successfully completed.
1 – The named DDT does not exist or is incomplete.
2 – Referenced DDT is not of type "history."
3 – Unable to access the LCN, or Data Link failed, or mailbox does not exist.
4 – Begin Offset is invalid.
5 – Request completed, but with errors.
6 – End Offset is invalid.
7 – History Type is not valid, or there was a Data Access failure during the history query.
9 – Could not obtain history data from the HM.
11 – Get History request timed out.
12 – Get History request queue is full.
13 – Begin/End offsets are invalid.
14 – Array size is too small.
15 – ACP connected ACIDP not in RUN state.
16 – Specified ACIDP not found.

See also, error codes listed in paragraphs A.2 and A.3.

G.4.3 History Data Return Formats

G.4.3.1 Averages Data Format

The normal averages data returned for each point specified in the History DDT is preceded by a 2-word header consisting of

Word 1—Data Access Status Code (16-bit Integer). See Appendix A for interpretation.

Word 2—Number of averages returned for this point (16-bit Integer).

This header is immediately followed by groups of 10 words for each returned average. The content of each average record varies according to its Value Type (word 1 of each 10-word record). There are two averages record forms:

Form 1—Word 1	=	0 (Normal data), or	
	=	1 (Nonstandard), or	
	=	5 (Time change), or	
	=	13 (Time change nonstandard)	(16-bit Integer)
Word 2..3	=	Average process value for period	(Real*)
Word 4..5	=	Time Stamp	(Array [1..2] of 16-bit Integers)
Word 6..7	=	Maximum process value in period**	(Real*)
Word 8..9	=	Minimum process value in period**	(Real*)
Word 10	=	Number of samples for period	(16-bit Integer)

Form 2—Word 1	=	6 (Outage), or	
	=	7 (No data), or	
	=	12 (Not in history)	(16-bit Integer)
Word 2..3	=	Unused (initialized to Integer -0)	
Word 4..5	=	Time Stamp	(Array [1..2] of 16-bit Integers)
Word 6..7	=	Unused (initialized to Integer -0)	
Word 8..9	=	Unused (initialized to Integer -0)	
Word 10	=	Unused	

Exceptions: If the Data Access Status code for a point (header word 1) is other than 4, 5, 6, or 8, there are no average values for that point and no other words are used. If the number of averages for a point is zero, only the two header words are used.

Record Word 1 (Value Type) definitions for averages data are

0	=	Normal: 90% or more good samples
1	=	Nonstandard: Less than 90% good samples
2	=	Digital Value: not applicable (if an average is requested for a parameter of type digital, the value type returned is "data not in history").
3-4	=	not used
5	=	Time Change: a time change occurred during the averaging period but there are 90% or more good samples.

*See paragraph G.4.4 for required format conversion information.

** Because of the storage method used, Minimum/Maximum process values can have up to 1% error (+1% for Maximum and -1% for Minimum).

6 = Outage: History Module was not in service for the entire period; value field contains NaN.
7 = No Data: no values were available from the Data Owner for entire period; value field contains NaN.
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN.
13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples.

G.4.3.2 Snapshot Data Format

The normal snapshot data returned for each point specified in the History DDT is preceded by a 2-word header consisting of

Word 1—Data Access Status Code (16-bit Integer). See Appendix A for interpretation.

Word 2—Number of snapshots returned for this point (16-bit Integer).

This header is immediately followed by groups of five words for each returned snapshot. The content of each snapshot record varies according to its Value Type (word 1 of each 5-word record). There are three snapshot record forms:

Form 1—Word 1	=	0 (Real value data)	(16-bit Integer)
Word 2..3	=	Process value	(Real*)
Word 4..5	=	Time Stamp	(Array [1..2] of 16-bit Integers)
Form 2—Word 1	=	2 (Digital value data)	(16-bit Integer)
Word 2	=	Digital process value	(16-bit Integer)
Word 3	=	Unused (initialized to Integer -0)	
Word 4..5	=	Time Stamp	(Array [1..2] of 16-bit Integers)
Form 3—Word 1	=	5 (Time change), or	
	=	6 (Outage), or	
	=	7 (No data), or	
	=	11 (Collection inhibited), or	
	=	12 (Not in history)	(16-bit Integer)
Word 2..3	=	Unused (initialized to Integer -0)	
Word 4..5	=	Time Stamp	(Array [1..2] of 16-bit Integers)

Exceptions: If the Data Access Status code for a point (header word 1) is other than 4, 5, 6, or 8, there are no snapshot values for that point. When the number of snapshots for a point is zero, only the two header words are used.

*See paragraph G.4.4 for required format conversion information

Record Word 1 (Value Type) definitions for Snapshot data are

0 = Normal Data: value returned is analog (real) data
1 = Nonstandard: not applicable
2 = Digital Value: value returned is a self-defined enumeration
3-4 = not used
5 = Time Change: a time change occurred and data for one minute is missing; value field contains NaN
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: History collection was not enabled; value field contains NaN
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: not applicable

G.4.4 History Data Format Conversion

Because values of type Real are returned by the Get History call, but the receiving array is an array of 16-bit Integer values, the ACP must convert adjacent pairs of integers into reals. Furthermore, it must provide for the detection of any bad values (see paragraph G.7.1) that may have been returned by the call. Bad-value detection must be done before any floating-point operations (including assignment) are performed on the data.

Following are sample FORTRAN and Pascal routines that examine a pair of 16-bit integers and return either a real value, or an indication that the integers represent a bad value.

Sample FORTRAN Conversion Routine

```
SUBROUTINE CONVIR (V1, V2, BADVAL, VALUE)
16-BIT INTEGER2 V1, V2
LOGICAL*2 BADVAL
REAL VALUE
LOGICAL*2 GOOD
16-BIT INTEGER2 I(2)
REAL R
EQUIVALENCE (I, R)
I(1) = V1
I(2) = V2
CALL CHKBAD (R, GOOD)
BADVAL = .NOT. GOOD
IF (GOOD) THEN
     VALUE = R
ELSE
     VALUE = 0.0
ENDIF
END
```

Sample Pascal Conversion Routine

```
PROCEDURE Convert_Integer_to_Real (V1,V2        :INTEGER_2_TYPE;
                                   VAR Badvalue :Boolean;
                                   VAR Value    :Real);

TYPE
  INTEGER_2_TYPE = [WORD] -32768..32767;
  Integer_Pair_Type = ARRAY [1..2] OF INTEGER_2_TYPE;

VAR
    Good: Boolean;
    Trix: [VOLATILE] PACKED RECORD
              CASE Boolean OF
                    True:  (I: Integer_Pair_Type);
                    False: (R_VALUE: Real);
              END;

BEGIN
WITH Trix DO
    BEGIN
    I[1] := V1;
    I[2] := V2;
    R := R_VALUE;
    ChkBad (R, Good);
    Badvalue := NOT Good;
    IF Good THEN
          Value := R
    ELSE
          Value := 0.0;
    END;
END;
```

G.5 TEXT MESSAGE TRANSFERS

The two interface routines in this group are used to send and receive character-string messages over the LCN.

G.5.1 Get Message Interface

This routine is used to fetch a character-string message held in a buffer by this program's ACIDP. The message presence is determined as the result of a Get ACP Status request.

G.5.1.1 Pascal Procedure Call for Get Message

```
GETMSG (%DESCR TEXT_ARRAY, TEXT_ARRAY_SIZE, RETURN_STATUS);
```

Note the required use of %DESCR with some parameters.

G.5.1.2 Typical Pascal Data Definition for Get Message

```
TYPE
   INTEGER_2_TYPE  : [WORD] -32768..32767;

VAR
   TEXT_ARRAY      : PACKED ARRAY [1..120] OF CHAR;
   TEXT_ARRAY_SIZE : INTEGER_2_TYPE;
   RETURN_STATUS   : INTEGER_2_TYPE;
```

The data definition shown for TEXT_ARRAY is its maximum.

G.5.1.3 FORTRAN Subroutine Call for Get Message

```
CALL GETMSF (TEXT_ARRAY, TEXT_ARRAY_SIZE, RETURN_STATUS)
```

G.5.1.4 Typical FORTRAN Data Definitions for Get Message

```
CHARACTER*120     TEXT_ARRAY
INTEGER*2         TEXT_ARRAY_SIZE
INTEGER*2         RETURN_STATUS
```

G.5.1.5 Parameter Definitions for Get Message

TEXT_ARRAY—The name of a packed array of characters where the message is to be stored.

TEXT_ARRAY_SIZE—The name of a 16-bit Integer variable that specifies the maximum number of characters you expect (120-character limit).

RETURN_STATUS—The name of a 16-bit Integer variable where the request-completion status is to be stored. The assigned meanings are

0 – Normal return—request was successfully completed.
1 – Request completed, but trailing characters in excess of Text Array Size have been deleted.
2– No message has been sent.
3– Unable to access the LCN.
4– Request was successfully completed. A second message is now queued at the ACIDP.
5 – Request completed with trailing characters deleted. A second message is now queued at the ACIDP.
6 – Valid ACIDP not found
8 – The specified number of characters to be received is negative or greater than 120.
15 – ACP connected ACIDP not in RUN state.

See also, error codes listed in paragraphs A.2, and A.3.

G.5.2 Send Message Interface

This routine is used to send a message to all operator stations assigned to the same unit as this program's ACIDP. A request to wait for operator confirmation is optional. If operator confirmation is requested, execution of the requesting program is suspended until either the confirmation occurs or until its specified wait time expires. The requesting program receives an indication of whether confirmation or a time out occurs.

G.5.2.1 Pascal Procedure Call for Send Message

```
SNDMSG (%DESCR TEXT_ARRAY, MESSAGE_SIZE, CONFIRM, WAIT_TIME,
        DESTINATION, RETURN_STATUS);
```

Note the required use of %DESCR with some parameters.

G.5.2.2 Typical Pascal Data Definition for Send Message

```
TYPE
   INTEGER_2_TYPE : [WORD] -32768..32767;
   BOOLEAN_2_TYPE : [WORD] BOOLEAN;

VAR
   TEXT_ARRAY     : PACKED ARRAY [1..120] OF CHAR;
   MESSAGE_SIZE   : INTEGER_2_TYPE;
   CONFIRM        : BOOLEAN_2_TYPE;
   WAIT_TIME      : INTEGER_2_TYPE;
   DESTINATION    : INTEGER_2_TYPE;
   RETURN_STATUS  : INTEGER_2_TYPE;
```

G.5.2.3 FORTRAN Subroutine Call for Send Message

```
CALL SNDMSF   (TEXT_ARRAY, MESSAGE_SIZE, CONFIRM, WAIT_TIME,
               DESTINATION, RETURN_STATUS)
```

G.5.2.4 Typical FORTRAN Data Definitions for Send Message

```
CHARACTER*120     TEXT_ARRAY
INTEGER*2         MESSAGE_SIZE
LOGICAL*2         CONFIRM
INTEGER*2         WAIT_TIME
INTEGER*2         DESTINATION
INTEGER*2         RETURN_STATUS
```

G.5.2.5 Parameter Definitions for Send Message

TEXT_ARRAY—The name of a packed array of characters that contains the message to be sent.

MESSAGE_SIZE—The name of a 16-bit Integer variable that specifies the number of characters to be transmitted. The maximum number of characters depends on message destination: 60 for CRT displays and 72 for printing. Over-length messages are truncated. All messages are archived if the HM is so configured.

CONFIRM—The name of a Boolean variable (TRUE or FALSE) that specifies whether or not a message confirmation is required. Note that this parameter is treated as FALSE if the message destination is printer only.

WAIT_TIME—The name of a 16-bit Integer variable (0 to 3600) that specifies the number of seconds the system is to wait for confirmation before returning control to the requesting program with a "no confirm" return status. (Allow for a built-in time lag of up-to-10 seconds.) The Wait Time parameter is ignored if the Confirm parameter is set to OFF or the message destination is printer only.

DESTINATION—The name of a 16-bit Integer variable that specifies where the message is to be sent, as follows:

0 – CRT only
1 – Printer only
2 – Both

RETURN_STATUS—The name of a 16-bit Integer variable where the request-completion status is to be stored. The assigned meanings are

0 – Normal return—request was successfully completed (and, if requested, confirmed).
2 – Wait Time not within allowed range, or message could not be sent.
3 – Unable to access the LCN.
4 – Message was sent, but wait time elapsed with no confirmation.
5 – Destination not within allowed range.
6 – Valid ACIDP not found.
7 – Confirm parameter not ON or OFF.
8 – The number of characters to be transmitted is negative.
15 – ACP connected ACIDP not in RUN state.

See also, error codes listed in paragraphs A.2. and A.3.

G.5.2.6 Event-Initiated Reports

Two types of Event-Initiated Reports can be invoked by specially formatted messages from an ACP to the Area Universal Stations:

- Logs, reports journals, and trends configured in the Area database.
- Event History reports.

Details of message requirements are given in Section 30 of the *Engineer's Reference Manual* located in the *Implementation/Startup & Reconfiguration - 2* binder.

G.6 ACP EXECUTION SUPPORT

These interface routines affect the orderly execution and termination of an ACP.

G.6.1 ACP Trap Handler Interface

This routine **must** be the first executable statement in each ACP. It establishes a termination handler for the ACP and ensures proper ACPI table setup. Failure to invoke this interface routine as the first statement of an ACP may not appear to cause immediate problems, but results in improper termination handling. The termination status is not reported to the CG, and the ACP appears to both the CM50S and the CG to still be in the RUN state even though the process has terminated.

G.6.1.1 Pascal Procedure Call for ACP Trap Handler

```
ACPTRP;
```

G.6.1.2 FORTRAN Subroutine Call for ACP Trap Handler

```
CALL ACPTRP
```

G.6.1.3 Data Definitions for ACP Trap Handler

Not applicable

G.6.2 Get ACP Status Interface

This routine fetches a set of parameters that enables the requesting program to determine why the system has turned it on and what special processing may be required at this time. It should be used during both the "setup" and "cleanup" program stages each time a Type 1 ACP runs. After servicing this request, the interface routine resets its copy of these values in preparation for any subsequent ACP turn on.

G.6.2.1 Pascal Procedure Call for Get ACP Status

```
GETSTS (TAKE_I_P, PS_MSG, DEMAND, PROCSPEC, SCHEDULED, UPPER_LEVEL);
```

G.6.2.2 Typical Pascal Data Definitions for Get ACP Status

```
VAR
   TAKE_I_P       : BOOLEAN_2_TYPE;
   PS_MSG         : BOOLEAN_2_TYPE;
   DEMAND         : BOOLEAN_2_TYPE;
   PROCSPEC       : BOOLEAN_2_TYPE;
   SCHEDULED      : BOOLEAN_2_TYPE;
   UPPER_LEVEL    : BOOLEAN_2_TYPE;
   BOOLEAN_2_TYPE : [WORD] BOOLEAN;
```

G.6.2.3 FORTRAN Subroutine Call for Get ACP Status

```
CALL GETSTS (TAKE_I_P, PS_MSG, DEMAND, PROCSPEC, SCHEDULED, UPPER_LEVEL)
```

G.6.2.4 Typical FORTRAN Data Definitions for Get ACP Status

```
LOGICAL*2         TAKE_I_P
LOGICAL*2         PS_MSG
LOGICAL*2         DEMAND
LOGICAL*2         PROCSPEC
LOGICAL*2         SCHEDULED
LOGICAL*2         UPPER_LEVEL
```

G.6.2.5 Parameter Definitions for Get ACP Status

TAKE_I_P—The name of a Boolean variable that returns TRUE the first time this program is turned on by the CG, following an initialization event . TAKE_I_P should be ignored when UPPER_LEVEL is TRUE.

PS_MSG—The name of a Boolean variable that returns TRUE if a message for the program is waiting at the CG.

DEMAND—The name of a Boolean variable that returns TRUE if the program was turned on as the result of a process operator request.

PROCSPEC—The name of a Boolean variable that returns TRUE if the program was turned on as the result of a process special to its ACIDP from an HG, AM, or another ACP.

SCHEDULED—The name of a Boolean variable that returns TRUE if the program was turned on by periodic or cyclic scheduling.

UPPER_LEVEL—The name of a Boolean variable that returns TRUE if the program was turned on by the ACP Installer.

G.6.3 ACP Delay Interface

This routine suspends execution of the calling program for a specified number of seconds. Program execution resumes at the statement following the PRGDLY call.

G.6.3.1 Pascal Procedure Call for ACP Delay

```
PRGDLY (DELAY_TIME, RETURN_STATUS);
```

G.6.3.2 Typical Pascal Data Definitions for ACP Delay

```
TYPE
   INTEGER_2_TYPE : [WORD] -32768..32767;

VAR
   DELAY_TIME     : INTEGER_2_TYPE;
   RETURN_STATUS  : INTEGER_2_TYPE;
```

G.6.3.3 FORTRAN Subroutine Call for ACP Delay

```
CALL PRGDLY   (DELAY_TIME, RETURN_STATUS)
```

G.6.3.4 Typical FORTRAN Data Definitions for ACP Delay

```
INTEGER*2         DELAY_TIME
INTEGER*2         RETURN_STATUS
```

G.6.3.5 Parameter Definitions for ACP Delay

DELAY_TIME—A 16-bit Integer variable (1 to 60) that contains the length of time, in seconds, that the requesting program is to be suspended.

RETURN_STATUS—A 16-bit Integer variable where the request completion status is to be stored. The assigned meanings are

0 – Request was successfully completed.
1 – Invalid delay time (not in 1-60 range).

G.6.4 ACP Termination Interface

This routine terminates the execution of the calling ACP. It must be used as the last operating statement of each ACP. If there is an associated ACIDP, a termination-status code is stored in its ABORTCOD parameter.

The termination code assigned by this call can be viewed at a Universal Station (see the definitions for ABORTCOD and EXECSTAT) but in a revised form. The integer value assigned here is translated into two hexadecimal digits (00 to FF) and appended to the character string EA. Thus, an ACP-assigned abnormal termination code of 15 appears at the Universal Station display as EA0F.

If an ACP is aborted by the VMS operating system, an abort code of UMSF is stored in its ACIDP's ABORTCOD.

The execution state of an ACIDP can be changed from ABORT to normal by operator demand through a Universal Station or by invoking the Deactivate/Terminate function of ACPI.

G.6.4.1 Pascal Procedure Call for ACP Termination

```
PRGTRM (TERMINATE_CODE);
```

G.6.4.2 Pascal Data Definition for ACP Termination

```
VAR
   TERMINATE CODE : INTEGER;
```

G.6.4.3 FORTRAN Subroutine Call for ACP Termination

```
CALL PRGTRM   (TERMINATE_CODE)
```

G.6.4.4 Typical FORTRAN Data Definitions for ACP Termination

```
INTEGER*4         TERMINATE_CODE
```

G.6.4.5 Parameter Definitions for ACP Termination

TERMINATE_ CODE—The name of a 32-bit Integer that must contain zero or a positive value (1 to 255). Zero value indicates normal termination. Nonzero values are user-specified codes for non-normal termination (abort).

G.7 UTILITY ROUTINES

The interface routines in this group provide support by the manipulation of data formats.

G.7.1 Check Bad Value Interface

This routine checks a value of type "real" to determine if it is a valid single-precision floating-point number. Its primary purpose is to check for the "Bad Value" indicator, NaN (-0).

G.7.1.1 Pascal Procedure Call for Check Bad Value

```
CHKBAD (VALUE, VALUE_STATUS);
```

G.7.1.2 Typical Pascal Data Definitions for Check Bad Value

```
VAR
   VALUE          : REAL;
   VALUE_STATUS   : BOOLEAN_2_TYPE;
   BOOLEAN_2_TYPE : [WORD] BOOLEAN;
```

G.7.1.3 FORTRAN Subroutine Call for Check Bad Value

```
CALL CHKBAD   (VALUE, VALUE_STATUS)
```

G.7.1.4 Typical FORTRAN Data Definitions for Check Bad Value

```
REAL*4            VALUE
LOGICAL*2         VALUE_STATUS
```

G.7.1.5 Parameter Definitions for Check Bad Value

VALUE—The name of a variable of type "real" that contains a single-precision floating-point value that is to be checked.

VALUE_STATUS—The name of a Boolean variable that returns TRUE if "Value" is found to be a valid floating-point number. It returns FALSE for minus zero (NaN) or other invalid bit configurations.

G.7.2 Set Bad Value Interface

This routine stores the bad value constant, NaN (-0), into the specified "real" variable.

G.7.2.1 Pascal Procedure Call for Set Bad Value

```
SETBAD (VARIABLE_NAME);
```

G.7.2.2 Typical Pascal Data Definitions for Set Bad Value

```
VAR
   VARIABLE_NAME : REAL;
```

G.7.2.3 FORTRAN Subroutine Call for Set Bad Value

```
CALL SETBAD   (VARIABLE_NAME)
```

G.7.2.4 Typical FORTRAN Data Definitions for Set Bad Value

```
REAL*4            VARIABLE_NAME
```

G.7.2.5 Parameter Definitions for Set Bad Value

VARIABLE_NAME—The name of a variable of type "real" (single-precision floating-point).

G.7.3 Convert External to Internal ID Interface

This routine fetches the internal ID of a point.parameter for the calling program. Use of the internal ID by repetitive single-value data gets and stores reduces system overhead and provides somewhat faster return of data. The specification of which point.parameter internal ID is wanted and where it is to be stored is contained in the call.

G.7.3.1 Pascal Procedure Call for Convert ID

```
SP_CVT (POINT_ID, POINT_ENT_INDEX, PT_PARAM, PARAM_INDEX,
        ID_BLOCK, VALUE_TYPE, PRIORITY, IUNIT, RETURN_STATUS);
```

G.7.3.2 Typical Pascal Data Definitions for Convert ID

```
TYPE
   INTEGER_2_TYPE  : [WORD] -32768..32767;

VAR
   POINT_ID        : PACKED ARRAY [1..8] OF CHAR;
   POINT_ENT_INDEX : INTEGER_2_TYPE;
   PT_PARAM        : PACKED ARRAY [1..8] OF CHAR;
   PARAM_INDEX     : INTEGER_2_TYPE;
   ID_BLOCK        : ARRAY [1..8] OF INTEGER_2_TYPE;
   VALUE_TYPE      : INTEGER_2_TYPE;
   PRIORITY        : INTEGER_2_TYPE;
   IUNIT           : INTEGER_2_TYPE;
   RETURN_STATUS   : INTEGER_2_TYPE;
```

G.7.3.3 FORTRAN Subroutine Call for Convert ID

```
CALL SPCVTF   (POINT_ID, POINT_ENT_INDEX, PT_PARAM, PARAM_INDEX,
               ID_BLOCK, VALUE_TYPE, PRIORITY, IUNIT, RETURN_STATUS)
```

G.7.3.4 Typical FORTRAN Data Definitions for Convert ID

```
CHARACTER*8       POINT_ID
INTEGER*2         POINT_ENT_INDEX
CHARACTER*8       PT_PARAM
INTEGER*2         PARAM_INDEX
INTEGER*2         ID_BLOCK(8)
INTEGER*2         VALUE_TYPE
INTEGER*2         PRIORITY
INTEGER*2         IUNIT
INTEGER*2         RETURN_STATUS
```

G.7.3.5 Parameter Definitions for Convert ID

POINT_ID—The name of a packed array of eight characters that contains the ASCII Point ID (name) of the point for which the internal ID is to be obtained.

POINT_ENT_INDEX—The name of a 16-bit Integer variable that contains an index for use with point arrays. This feature is not implemented and the value must be zero.

PT_PARAM—The name of a packed array of eight characters that contains the ASCII parameter name for the point.parameter for which the internal ID is to be obtained.

PARAM_INDEX—The name of a 16-bit Integer variable whose interpretation is controlled by VALUE_TYPE.

When VALUE_TYPE is 1, 2, 3, 4, or 5, a single value is to be accessed; this can be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of PARAM_INDEX is used as an index and must be greater than zero. If the parameter being accessed is not an array type, the PARAM_INDEX value must be zero.

When VALUE_TYPE is 7, 8, 9, or 10, a whole array is to be accessed and PARAM_INDEX is used to specify the expected number of elements in the array. Its value must be greater than zero and not larger than 1000. The actual size of the array is returned in the seventh word of the 8-word ID_BLOCK. If the PARAM_INDEX value is ≤0 or >1000, RETURN_STATUS equals 10.

ID_BLOCK—The name of a 16-bit Integer array into which the internal ID data block is to be stored. Save these eight values for later use in SPIGTF or SPISTF calls on this point.parameter. See paragraph 4.7.8 of the *Computer Gateway User Manual* for ID block details.

VALUE_TYPE—The name of a 16-bit Integer variable that contains a number that designates value type as follows:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
6 = not used
7 = Real array
8 = Integer array
9 = Enumeration array
10 = Ordinal value of enumeration array

PRIORITY—The name of a 16-bit Integer variable that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

IUNIT—The name of a 16-bit Integer variable (1-4) identifying the CG to be accessed.

RETURN_STATUS—The name of a 16-bit Integer variable where the request-completion status is to be stored. The assigned meanings are

0 – Normal return—request successfully completed.
3 – Unable to access the LCN, or data link failed, or mailbox does not exist.
5 – Request had errors.
6 – Point entity or parameter not found on the LCN.
9 – Illegal priority number (must be 1 or 2).
10 – Invalid call parameters or the array size is ≤0 or >1000.
11 – Invalid data type or data type does not match the parameter type.
13 – CG message transaction error—internal error.
14 – Parameter Index is smaller than array size; request was completed successfully
15 – Point.parameter ID or ASCII/Enumeration value not on word boundary.
22 – Data type out of range
27 – Index out of range
28 – Bad DA priority
29 – Bad data type
30 – Bad entity index
32 – Bad parameter index
40 – Number of words specified in message header disagrees with message content.
41 – Parameter Index is greater than the array size

See also, error codes listed in paragraphs A.2, and A.3.

G.7.4 Convert LCN Time (Time Stamp)

This routine converts LCN time (a count in seconds since January 1, 1979) to external time. It accepts the LCN-time value (typically received as time-stamp data in response to a Get History request) and returns external time in the form MM/DD/YYΔHH:MM.

G.7.4.1 Pascal Procedure Call for Convert LCN Time

```
CONVTP (V1, V2, ASCTIME);
```

G.7.4.2 Typical Pascal Data Definitions for Convert LCN Time

```
TYPE
   INTEGER_2_TYPE  : [WORD] -32768..32767;

VAR
   V1              : INTEGER_2_TYPE;
   V2              : INTEGER_2_TYPE;
   ASCTIME         : PACKED ARRAY [1..14] OF CHAR;
```

G.7.4.3 FORTRAN Subroutine Call for Convert LCN Time

```
CALL CONVTF   (V1,V2,ASCTIME)
```

G.7.4.4 Typical FORTRAN Data Definitions for Convert LCN Time

```
INTEGER*2         V1
INTEGER*2         V2
CHARACTER*14      ASCTIME
```

G.7.4.5 Parameter Definitions for Convert LCN Time

V1—The name of a 16-bit Integer variable that contains the upper-half of a value representing LCN time.

V2—The name of a 16-bit Integer variable that contains the lower-half of a value representing LCN time.

ASCTIME—The name of a character-string variable where the converted time is to be stored.

G.8 SINGLE POINT DATA TRANSFERS

The interface routines in this group Get or Store values from or to one named point.parameter (or parameter array) at a time. For parameter arrays, up-to the whole array is accessed. The External ID version of Get Single Point is also used to get LCN date and time.

G.8.1 Get Single Point (External ID) Interface

This routine fetches data for a single point from a specified CG or elsewhere on its LCN. The specification of which data is to be fetched and where it is to be stored is contained in the call. For parameter arrays, either a single element, the whole array, or an array subset starting with the first element can be specified.

G.8.1.1 Example FORTRAN Call for Get Single Point

```
return_status =  CM50_GETPT
                 (%REF(point),
                  %REF(param),
                  param_ix,
                  %REF(val_loc),
                  val_st,
                  val_typ,
                  cg_port_num)
```

G.8.1.2 Parameter Definitions for Get Single Point

return_status—The name of a longword to receive the overall return status of the function call. See Appendix A.2 for an explanation and a listing of all assigned return code values. The most common return status values from this call follow.

return_ status Value	Identifier	Include-File Error Message
000000001	SS$_NORMAL	<none>
215000036	CM50_LCN_FAIL	Unable to access LCN—data link failure
215000042	CM50_ACP_RUN	ACIDP not in RUN state
215000051	CM50_LCN_PART	Returned Data includes errors
215000066	CM50_LCN_POINT	ACIDP not found or not connected to ACP
215000092	CM50_LCN_PRIOR	Invalid Priority—must be 1 or 2
215000106	CM50_LCN_PARAM	Point or Parameter not found
215000116	CM50_LCN_PARTYP	Value Type does NOT match Parameter Type
215000122	CM50_GET_MEM	Unable to Get Memory in VAX
215000156	CM50_LCN_WORD	Value not Word Aligned
215000226	CM50_LCN_RANGE	Data Type Out of Range
215000292	CM50_LCN_TYPE	Invalid data Type code
215000322	CM50_ACC_SIZE	Array pointer/size too Large

point—The name of an 8-character string that contains the ASCII Point ID (name) of the point from which the value is to be retrieved.

param—The name of an 8-character string that contains the ASCII name of a parameter (or parameter array) from which the value(s) is retrieved.

param_ix—The name of a shortword positive value. Use of this value is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, or 5, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of `param_ix` is used as an index and must be greater than zero. If the parameter being accessed is not an array type, the `param_ix` value must be zero.

When `val_typ` is 7, 8, 9, or 10, a whole array (or a subset of the array starting with the first element) is to be accessed and `param_ix` is used to specify the number of elements to be accessed—If `param_ix` is smaller than the actual array size, only that number of elements is returned; if it is larger than the actual array size, `return_status` equals `CM50_LCN_ARRAY` (no elements are returned).

val_loc—The name of a program variable where the value(s) are to be stored. The type of variable must match what is declared in `val_typ`. (Pascal programs must use explicit pass by reference.)

val_typ	val_loc type
1	Real (32 bits)
2	Integer (shortword)
3	ASCII_VALS (24-character string)
4	ENUM_VALS (8-character string)
5	Integer (shortword)
6	TIME_ARR (18-character string, see heading G.8.3)
7	ARRAY [1..n] OF REAL (32 bit values)
8	ARRAY [1..n] OF INTEGER (shortwords)
9	ARRAY [1..n] OF ENUM_VALS (8-character strings)
10	ARRAY [1..n] OF INTEGER (shortwords)

val_st—The name of a shortword where point-related status information is to be stored. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (215000051). See Appendix A.1 for a listing of Data-Access error/status codes. When `val_typ` specifies an array, `val_st` refers to status of the whole array.

val_typ—The name of a shortword that contains a number that designates value type of the accessed parameters as follows:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
6 = LCN time (see heading G.8.3)
7 = Real array
8 = Integer array
9 = Enumeration array
10 = Ordinal value of enumeration array

cg_port_num—The name of a shortword identifying the CG (1-4) to be accessed.

G.8.2 Store Single Point (External ID) Interface

This routine stores data to a single point in a specified CG or elsewhere on its LCN. The specification of where the data is to be found and where it is to be stored is contained in the call. For parameter arrays, either a single element or the whole array can be specified.

To use this call, the ACP must be connected to an ACIDP with read/write access and be in Normal mode. See the *System Control Functions* manual for other write access restrictions.

G.8.2.1 Example FORTRAN Call for Store Single Point

```
return_status =  CM50_STOREPT
                 (%REF(point),
                  %REF(param),
                  param_ix,
                  %REF(val_loc),
                  val_typ,
                  store_cd,
                  store_st)
```

G.8.2.2 Parameter Definitions for Store Single Point

return_status—The name of a longword to receive the overall return status of the function call. See Appendix A.2 for an explanation and a listing of all assigned return code values. The most common return status values from this call follow.

return_ status Value	Identifier	Include-File Error Message
000000001	SS$_NORMAL	<none>
215000012	CM50_DDT_MISS	DDT incomplete or not found
215000036	CM50_LCN_FAIL	Unable to access LCN—data link failure
215000042	CM50_ACP_RUN	ACIDP not in RUN state
215000051	CM50_LCN_PART	Returned Data includes errors
215000066	CM50_LCN_POINT	ACIDP not found or not connected to ACP
215000076	CM50_ACP_STAT	ACP not installed or restricted
215000082	CM50_ACP_ACCE	ACCESS key is Read only or DDT is in use
215000106	CM50_LCN_PARAM	Point or Parameter not found
215000116	CM50_LCN_PARTYP	Value Type does NOT match Parameter Type
215000122	CM50_GET_MEM	Unable to Get Memory in VAX
215000146	CM50_LCN_ARRAY	Array Size too small
215000156	CM50_LCN_WORD	Value not Word Aligned
215000226	CM50_LCN_RANGE	Data Type Out of Range
215000292	CM50_LCN_TYPE	Invalid data Type code
215000306	CM50_LCN_PTIX	Point Index not Zero
215000322	CM50_ACC_SIZE	Array pointer/size too Large

point—The name of an 8-character string that contains the ASCII Point ID (name) of the point where the value is to be stored.

param—The name of 8-character string that contains the ASCII parameter name for the point.parameter where the value is to be stored.

param_ix—The name of a shortword positive value. Use of this value is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, or 5, a single value is to be accessed—This may be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of `param_ix` is used as an index and must be greater than zero. If the parameter being accessed is not an array type, the `param_ix` value must be zero.

When `val_typ` is 7, 8, 9, or 10, a whole array is to be accessed and `param_ix` is used to specify the number of array elements—If `param_ix` does not match the actual array size, no elements are stored and `return_status` value is `CM50_LCN_PART` with a `store_st` indicating an invalid array size.

val_loc—The name of a program variable from which the value(s) are to be fetched. The type of variable must match what is declared in `val_typ`. (Pascal programs must use explicit pass by reference.)

val_typ	val_loc type
1	Real (32 bits)
2	Integer (shortword)
3	ASCII_VALS (24-character string)
4	ENUM_VALS (8-character string)
5	Integer (shortword)
6	not used
7	ARRAY [1..n] OF REAL (32 bit values)
8	ARRAY [1..n] OF INTEGER (shortwords)
9	ARRAY [1..n] OF ENUM_VALS (8-character strings)
10	ARRAY [1..n] OF INTEGER (shortwords)

val_typ—The name of a shortword that contains a number that designates value type as follows:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
6 = not used
7 = Real array (conversion at CG)
8 = Integer array
9 = Enumeration array
10 = Ordinal value of enumeration array

store_cd—Name of a shortword that contains a code that allows the substitution of a bad value representation in place of the provided value(s). The store code values are

0 = Store the data value(s) provided
1 = Store the bad value representation instead

Store code 1 is valid for only Real or ASCII data. The bad value representations are NaN for Real values and question mark strings for ASCII.

store_st—The name of a shortword to contain point-related store status information on completion. This value is meaningful only when the `return_status` value indicates either normal (000000001) or complete with errors (215000051). See Appendix A.1 for a listing of Data-Access error/status codes. When the `val_typ` is an array, `store_st` refers to status of the whole array.

G.8.3 Get LCN Clock Value Interface

To get the current date and time as kept by the LCN, use the Get Single Point (External ID) interface using the following definitions or values:

`val_loc`—type must be character array (CHARACTER*18)
`val_typ` = 6
`cg_port_num` = CG port number for the LCN (1-4)

The other arguments, including point and parameter, should be passed as blanks. The format of the value returned in `val_loc` is MM/DD/YYΔHH:MM:SSΔ (where Δ is used to indicate a space).

See heading G.8.1 for Get Single Point call details.

G.9 POINT ARRAYS WITHOUT DDTS

These routines enable you to address multiple points with a single call without the necessity to build DDT tables. In the place of a DDT reference, you will have to provide a pointer to an array of "internal" point.parameter addresses. These internal addresses can be obtained by a series of CM50_CONV_ID (or _TP) calls at program run time, or in advance by creating an include file through the Utility MAKEINC.

G.9.1 Point List Get Values Interfaces

These functions return data values from up to 300 points on the LCN without using DDT tables. The specification of which data is to be fetched and where it is to be stored is contained in the call.

Use of Internal point.parameter IDs is required. Individual elements of parameter arrays can be specified by repeating the point.parameter address using a changed parameter index.

G.9.1.1 Example FORTRAN Calls for Point List Get Values

for REAL values:

```
return_status =  CM50_GET_REAL
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  real_values_array,
                  status_table,
                  number_of_values)
```

for INTEGER values:

```
return_status =  CM50_GET_INTG
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  intg_values_array,
                  status_table,
                  number_of_values)
```

for ASCII values:

```
return_status =  CM50_GET_ASCI
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(asci_values_array),
                  status_table,
                  number_of_values)
```

for ENUMERATED values:

```
return_status =  CM50_GET_AENM
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(aenm_values_array),
                  status_table,
                  number_of_values)
```

for ORDINAL values

```
return_status =  CM50_GET_OENM
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  oenm_values_array,
                  status_table,
                  number_of_values)
```

G.9.1.2 Common Parameter Definitions for Point List Get Values

return_status—The name of a longword to receive the overall return status of the function call. See Appendix A.2 for an explanation and a listing of all assigned return code values. The most common return status values from this call follow.

return_ status Value	Identifier	Include-File Error Message
000000001	SS$_NORMAL	<none>
215000036	CM50_LCN_FAIL	Unable to access LCN - data link failure
215000042	CM50_ACP_RUN	ACIDP not in RUN state
215000051	CM50_LCN_PART	Returned Data includes errors
215000066	CM50_LCN_POINT	ACIDP not found or not connected to ACP
215000092	CM50_LCN_PRIOR	Invalid Priority—must be 1 or 2
215000146	CM50_LCN_ARRAY	Array Size too small
215000906	CM50_LAX_ARGRNG	Argument out of Range

cg_port_num—The name of a shortword identifying the CG (1-4) to be accessed.

priority—The name of a shortword that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

acidp_name—The name of an 8-character string that names the ACIDP that the calling ACP (Type 1 program) is connected to. Left blank when the calling program is not connected to an ACIDP (Type 2 program) or you do not want the CG to control data gathering by a Type 1 program. When this argument is filled in, the CG requires the named ACIDP to show EXECSTAT = RUN and INH_STAT = PERMIT.

point_list_array—The name of an array of point addresses in internal format (for FORTRAN declare as `ARRAY OF RECORD/ID_BLOCK_STRUCT/`, or for Pascal declare as `ARRAY OF CM50_IDBLK`) from which the values are requested. See the Convert External to Internal ID functions for additional information.

This must be a maximum-size array even if it is not fully used. Use the constant CM50_MAXTBL to size this array.

status_table—The name of a shortword array of CM50_MAXTBL size where the value status for individual point values are to be stored. See Appendix A.1 for a listing of Data Access error/status codes. These additional values can replace "Data item is valid" (code 008):

CM50_Out_Of_VAX_Range
CM50_Negative_Infinity
CM50_Positive_Infinity
CM50_Not_A_Number
CM50_Minus_Zero
CM50_Underflow

Use these symbolic identifiers (not their assigned numeric values which are defined in the language-specific include files) in program statements since the numeric values could change in future releases of CM50S.

number_of_values—The name of a shortword that specifies the actual number of values (300 or less) to be processed.

G.9.1.3 Value Parameters for Each Data Type

real_values_array—The name of a Real array where the individual point values are to be stored. This array must be of CM50_MAXTBL size.

intg_values_array—The name of a shortword array where the individual point values are to be stored. This array must be of CM50_MAXTBL size.

asci_values_array—The name of an array of 24-character strings where the individual point values are to be stored. This array must be of CM50_MAXTBL size.

aenm_values_array—The name of an array of 8-character strings where the individual point values are to be stored. This array must be of CM50_MAXTBL size.

oenm_values_array—The name of a shortword array where the ordinal values of the fetched enumerations are to be stored. This array must be of CM50_MAXTBL size.

G.9.2 Point List Store Values Interfaces

These functions export data values to up to 300 points on the LCN without using DDT tables. The specification of which data is to be fetched and where it is to be stored is contained in the call.

Use of Internal point.parameter IDs is required. Individual elements of parameter arrays can be specified by repeating the point.parameter address using a changed parameter index.

G.9.2.1 Example FORTRAN Calls for Point List Store Values

for REAL values:

```
return_status =  CM50_STORE_REAL
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  real_values_array,
                  store_code_table,
                  status_table,
                  number_of_values)
```

for INTEGER values:

```
return_status =  CM50_STORE_INTG
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  intg_values_array,
                  store_code_table,
                  status_table,
                  number_of_values)
```

for ASCII values:

```
return_status =  CM50_STORE_ASCI
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(asci_values_array),
                  store_code_table,
                  status_table,
                  number_of_values)
```

for ENUMERATED values:

```
return_status =  CM50_STORE_AENM
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  %REF(aenm_values_array),
                  store_code_table,
                  status_table,
                  number_of_values)
```

for ORDINAL values:

```
return_status =  CM50_STORE_OENM
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(point_list_array),
                  oenm_values_array,
                  store_code_table,
                  status_table,
                  number_of_values)
```

G.9.2.2 Common Parameter Definitions for Array Store Values

return_status—The name of a longword to receive the overall return status of the function call. See Appendix A.2 for an explanation and a listing of all assigned return code values. The most common return status values from this call follow.

return_ status Value	Identifier	Include-File Error Message
000000001	SS$_NORMAL	<none>
215000036	CM50_LCN_FAIL	Unable to access LCN - data link failure
215000042	CM50_ACP_RUN	ACIDP not in RUN state
215000051	CM50_LCN_PART	Returned Data includes errors
215000066	CM50_LCN_POINT	ACIDP not found or not connected to ACP
215000082	CM50_ACP_ACCE	ACCESS key is Read only or DDT is in use
215000092	CM50_LCN_PRIOR	Invalid Priority—must be 1 or 2
215000146	CM50_LCN_ARRAY	Array Size too small
215000906	CM50_LAX_ARGRNG	Argument out of Range

cg_port_num—The name of a shortword identifying the CG (1-4) to be accessed.

priority—The name of a shortword that contains the requested data-access priority:

1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

acidp_name—The name of an 8-character string that names the ACIDP that the calling ACP (Type 1 program) is connected to. Required for LCN stores.

point_list_array—The name of an array of point addresses in internal format (for FORTRAN declare as `ARRAY OF RECORD/ID_BLOCK_STRUCT/`, or for Pascal declare as `ARRAY OF CM50_IDBLK`) from which the values are requested. See the Convert External to Internal ID functions for additional information.

This must be a maximum-size array even if it is not fully used. Use the constant CM50_MAXTBL to size this array.

store_code_table—The name of a shortword array where the calling program has stored a control code for each referenced variable in the `real_values_array`. This array must be of CM50_MAXTBL size. The `store_code_table` values are

0 = Store the referenced Real variable
1 = Store the bad value representation (NaN) instead
2 = Do not store any value

status_table—The name of a shortword array where the value status for each individual point value is to be stored. This array must be of CM50_MAXTBL size. See Appendix A.1 for interpretation of values.

number_of_values—The name of a shortword that specifies the actual number of values (300 or less) to be processed.

G.9.2.3 Value Parameters for Each Data Type

real_values_array—The name of a Real array from which the individual values are to be obtained. This array must be of CM50_MAXTBL size.

intg_values_array—The name of a shortword array from which the individual values are to be obtained. This array must be of CM50_MAXTBL size.

asci_values_array—The name of an array of 24-character strings from which the individual point values are to be obtained. This array must be of CM50_MAXTBL size.

aenm_values_array—The name of an array of 8-character strings from which the individual point values are to be obtained. This array must be of CM50_MAXTBL size.

oenm_values_array—The name of a shortword array from which the individual point values are to be obtained. This array must be of CM50_MAXTBL size.

G.10 SINGLE-POINT HISTORY CALLS

G.10.1 Single-Point Get History Interfaces

There are three versions of Single-Point Get History—Generalized, Real values, and Integer values.

- The generalized version handles both Relative history and Absolute history calls.
- The Real values version is used to obtain type Real data only and obtains history data by using an offset relative to an absolute (specified) time (which can be the current time).
- The Integer values version is used to obtain 1-minute snapshots of Integer values representing states of a self-defined enumeration, starting at an offset relative to a specified time (which can be the current time).

All three versions return historized data values for a point.parameter on the LCN without using DDT tables. The specification of which data is to be fetched and where it is to be stored is contained in the call.

Use of Internal point.parameter IDs (obtained by previous use of the Convert External to Internal ID interface) is required. Any parameter that can be historized, including an individual element of a parameter array, can be selected.

G.10.1.1 Example FORTRAN Calls for Single-Point Get History

for the Generalized version:

```
return_status =  CM50_GET_HIS
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(id_block),
                  history_mode,
                  history_type,
                  %REF(begin_date_time),
                  %REF(end_date_time),
                  begin_offset,
                  end_offset,
                  number_of_values,
                  real_values_array,
                  intg_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  value_status)
```

for the Real values version:

```
return_status =  CM50_GET_REAL_HIS
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(id_block),
                  history_type,
                  seconds_in_units,
                  %REF(common_start_time),
                  offset,
                  number_of_values,
                  real_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  max_array,
                  min_array,
                  num_samples_array,
                  value_status)
```

for the Integer values version:

```
return_status =  CM50_GET_INTG_HIS
                 (cg_port_num,
                  priority,
                  %REF(acidp_name),
                  %REF(id_block),
                  %REF(common_start_time),
                  offset,
                  number_of_values,
                  intg_values_array,
                  status_table,
                  lcn_time_stamp_array,
                  value_status)
```

G.10.1.2 Parameter Definitions for Single-Point Get History Calls

return_status—The name of a longword to receive the overall return status of the function call. See Appendix A.2 for an explanation and a listing of all assigned return code values. The most common return status values from this call follow.

return_ status Value	Identifier	Include-File Error Message
000000001	SS$_NORMAL	<none>
215000036	CM50_LCN_FAIL	Unable to access LCN—data link failure
215000042	CM50_ACP_RUN	ACIDP not in RUN state
215000051	CM50_LCN_PART	Returned Data includes errors
215000066	CM50_LCN_POINT	ACIDP not found or not connected to ACP
215000092	CM50_LCN_PRIOR	Invalid Priority—must be 1 or 2
215000146	CM50_LCN_ARRAY	Array Size too small
215000282	CM50_ACC_PRIOR	Invalid data Access Priority
215000521	CM50_MSG_TRUNC	Message truncated
215000802	CM50_LAX_MONV	Invalid Month in VAX date
215000842	CM50_LAX_NVAL	Invalid Number of Values requested
215000852	CM50_LAX_UNIT	Invalid History Units specified
215000906	CM50_LAX_ARGRNG	Argument out of Range
215000916	CM50_LAX_MONTHR	Month out of Range
215000922	CM50_LAX_BADOFF	Invalid History Offset
215000932	CM50_LAX_HISTYP	Invalid History Type

cg_port_num—The name of a shortword identifying the CG (1-4) to be accessed.

priority—The name of a shortword that contains the requested data-access priority:
1= High priority (provided for control operations)
2= Low priority (provided for noncontrol operations)

acidp_name—The name of an 8-character string that names the ACIDP that the calling ACP (Type 1 program) is connected to. Left blank when the calling program is not connected to an ACIDP (Type 2 program), or you do not want the CG to control data gathering by a Type 1 program. When this argument is filled in, the CG requires the named ACIDP to show EXECSTAT = RUN and INH_STAT = PERMIT.

id_block—The name of a 16-byte variable (for FORTRAN declare as `RECORD/ID_BLOCK_STRUCT/`, or for Pascal declare as `CM50_IDBLK`) that contains the internal ID data block of the point and parameter that history is being requested for. See the Convert External to Internal ID interface for additional information.

history_mode—The name of a shortword value that specifies an Absolute (0) or Relative (1) history request.

history_type—The name of a shortword that contains the number specifying the requested history type. The history types and their integer representations are

1-minute snapshots = 0
Hourly averages = 1
Shift averages = 2
Daily averages = 3
Monthly averages = 4
User averages = 5

seconds_in_units—The name of a shortword value that indicates the number of seconds in the history interval. Used only for Shift averages (History type 2) or User averages (History type 5); ignored for all other History types. Default values are: Shift averages = 8 hours (28800 seconds)
User averages = 6 minutes (360 seconds)

NOTE

When entering date and time in VMS absolute format, be sure to enter the 3-character code for the month in uppercase letters.

begin_date_time—The name of a 22-character string (in VMS "absolute" date and time format), representing the date and time starting the period for which history is to be fetched. For Relative history calls, this field has no meaning and should be initialized to blanks.

The value entered for this parameter should be some time after the previous period's collection, but before the desired time period begins. For example, to get an hourly average at 11:00, begin_date_time should be set any time between 10:01 and 10:59.

end_date_time—The name of a 22-character string (in VMS "absolute" date and time format), representing the date and time ending the period for which history is to be fetched. For Relative history calls, this field has no meaning and should be initialized to blanks.

The value entered for this parameter should be some time after the desired time period begins, but before the next time period's collection time. For example, end_date_time for an 11:00 hourly average should be set to any time from 11:01 to 11:59.

common_start_time—The name of a 23-character string that contains a beginning date and time (in VMS "absolute" date and time format) for history data collection.

begin_offset—The name of a shortword that indicates a relative offset from current LCN time that represents the starting period for which history is to be fetched. The units for the offset are based on the type of history requested.

HISTORY TYPE	UNITS OF OFFSET
1-minute snapshots	Minutes
Hourly averages	Hours
Shift averages	not applicable
Daily averages	Days
Monthly averages	Months
User averages	Number of user averages to skip

For Absolute history calls, this field has no meaning and should be initialized to zero.

end_offset—The name of a shortword that indicates a relative offset from the current LCN time representing the ending period for which history is to be fetched. The units for the offset are based on the type of history requested. See values given with BEGIN_OFFSET.

For Absolute history calls, this field has no meaning and should be initialized to zero.

offset—The name of a shortword value that establishes a beginning offset (expressed as a number of time units) to be subtracted from COMMON_START_TIME. The units used for this value are established by the History type as follows:

HISTORY TYPE	UNITS OF OFFSET
1-minute snapshots	Minutes
Hourly averages	Hours
Shift averages	(see SECONDS_IN_UNITS argument)
Daily averages	Days
Monthly averages	Months
User averages	(see SECONDS_IN_UNITS argument)

number_of_values—The name of a shortword that contains the number of values expected to be returned by the history request (250 or fewer). This variable is used to return to the caller, the actual number of values returned by the CG whenever the actual number is less than the number of values requested; otherwise this value is unchanged. If the actual number of values returned by the CG exceeds this limit, then the message is truncated at the limit.

For Absolute history requests, this number should be (at least) the number of intervals between the Beginning and Ending Date Times (note that it is easy to be off by one, depending on starting and stopping boundaries).

For Relative history requests, this number can be initialized as zero, but is checked as explained above. If provided, this number should be the absolute value of (BEGIN_OFFSET - END_OFFSET) +1.

real_values_array—The name of a Real array where the requested history data values are to be stored if the parameter's data type is Real.

intg_values_array—The name of a shortword array where the requested history data values are to be stored if the parameter's data type is Integer.

status_table—The name of a shortword array to receive a value status associated with each returned history data value. The value meanings are

For averages data:

0 = Normal: 90% or more good samples
1 = Nonstandard: Less than 90% good samples
2 = Digital Value: not applicable (if an average is requested for a parameter of type digital, the value type returned is "data not in history")
3-4 = not used
5 = Time Change: a time change occurred during the averaging period, but there are 90% or more good samples
6 = Outage: History Module was not in service for the entire period; value field contains NaN
7 = No Data: no values were available from the Data Owner for entire period; value field contains NaN
8-10 = not used
11 = Collection Inhibited: not applicable
12 = Not in History: requested data was outside span of the history file; value field contains NaN

13 = Time Change nonstandard: the average calculation was performed according to the new time, and samples already collected are rolled into the new average; there are fewer than 90% good samples

For Snapshot data:

0 = Normal Data: value returned is analog (real) data
1 = Nonstandard: not applicable
2 = Digital Value: value returned is a self-defined enumeration
3-4 = not used
5 = Time Change: a time change occurred and data for one minute is missing; value field contains NaN
6 = Outage: History Module was not in service; value field contains NaN
7 = No Data: the Data Owner was not in service; value field contains NaN
8-10 = not used
11 = Collection Inhibited: History collection was not enabled; value field contains NaN
12 = Not in History: requested data was outside span of the history file; value field contains NaN
13 = Time Change nonstandard: not applicable

For Real Values Only

These additional status values can replace Normal (code 0) for Real values in either averages or snapshots.

CM50_Out_Of_VAX_Range
CM50_Negative_Infinity
CM50_Positive_Infinity
CM50_Minus_Zero
CM50_Underflow

Use these identifiers (not their assigned numeric values which are defined in the language-specific include files) in your program statements, since the numeric values could change in future releases of CM50S.

lcn_time_stamp_array—The name of a longword array to receive a time stamp (count of the number of seconds since January 1, 1979) associated with each returned history data value.

max_array—(Averages only) The name of a Real array to receive the maximum values* used in calculating each returned Real_Values_Array average.

min_array—(Averages only) The name of a Real array to receive the minimum values* used in calculating each returned Real_Values_Array average.

num_samples_array—(Averages only) The name of a shortword array to receive the number of samples used in calculating each returned average value.

value_status—The name of a shortword to receive the Data Access Status Code returned from the CG. See Appendix A.1 for interpretation.

*Because of the storage method used, Minimum/Maximum process values can have up to 1% error (+1% for Maximum and -1% for Minimum).

G.11 ACP EXECUTION SUPPORT

These interface routines affect the orderly execution and termination of an ACP.

G.11.1 ACP Trap Handler Interface

This routine **must** be the first executable statement in each ACP. It establishes a termination handler for the ACP and ensures proper ACP table setup. Failure to invoke this interface routine as the first statement of an ACP may not appear to cause immediate problems, but will result in improper termination handling. The termination status is not reported to the CG, and the ACP appears to both the CM50S and the CG to still be in the RUN state even though the process has terminated.

The call to ACPTRP also establishes a system lock that allows the program to be terminated cleanly if CM50S is shut down. Therefore, it is advisable to include this call in every program that is mapped to the CM50S shareable image.

NOTE

ACPTRP is one of the few CM50S user-interface routines that is not implemented as a function. It is called as a FORTRAN subroutine or as a Pascal procedure.

G.11.1.1 Example Subroutine Call for ACP Trap Handler

```
CALL ACPTRP
```

G.11.1.2 Parameter Definitions for ACP Trap Handler

None required

G.12 UTILITY ROUTINES

The interface routines in this group provide support by the manipulation of data formats.

G.12.1 Convert External to Internal ID Interface

This routine fetches the internal ID of a point.parameter for the calling program. Use of the internal ID by repetitive single-value data gets and stores reduces system overhead and provides somewhat faster return of data. The specification of which point.parameter internal ID is wanted and where it is to be stored is contained in the call.

NOTE

The arrays of internal point.parameter addresses need to be rebuilt and the program(s) using them need to be recompiled, whenever the LCN database is changed in a significant manner, such as by the rebuild or deletion of data points referenced in the address array.

G.12.1.1 Example FORTRAN Call for Convert ID

```
return_status =  CM50_CONV_ID  or  CM50_CONV_TP
                 (%REF(point),
                  %REF(param),
                  param_ix,
                  %REF(id_block),
                  val_typ,
                  cg_port_num)
```

CM50_CONV_ID requires you to specify `val_typ`. CM50_CONV_TP tries different value type codes until it finds a match, then returns the actual `val_typ` code.

G.12.1.2 Parameter Definitions for Convert ID

return_status—The name of a longword to receive the overall return status of the function call. See Appendix A.2 for an explanation and a listing of all assigned return code values. The most common return_status values from this call follow.

return_ status Value	Identifier	Include-File Error Message
000000001	SS$_NORMAL	<none>
215000036	CM50_LCN_FAIL	Unable to access LCN—data link failure
215000106	CM50_LCN_PARAM	Point or Parameter not found
215000116	CM50_LCN_PARTYP	Value Type does NOT match Parameter Type
215000146	CM50_LCN_ARRAY	param_ix smaller than array size; conversion ok
215000156	CM50_LCN_WORD	Value not Word Aligned
215000226	CM50_LCN_RANGE	Data Type Out of Range
215000276	CM50_LCN_INDX	Index Out of Range
215000292	CM50_LCN_TYPE	Invalid data Type code

point—The name of an 8-character string that contains the ASCII Point ID (name) of the point for which the internal ID is to be obtained.

param—The name of an 8-character string that contains the ASCII parameter name for the point.parameter for which the internal ID is to be obtained.

param_ix—The name of a shortword whose interpretation is controlled by `val_typ`.

When `val_typ` is 1, 2, 3, 4, or 5, a single value is to be accessed; this can be an element of a parameter array (except for ASCII values). If the parameter to be accessed is an array type, the value of `param_ix` is used as an index and must be greater than zero. If the parameter being accessed is not an array type, the `param_ix` value must be zero.

When `val_typ` is 7, 8, 9, or 10, a whole array is to be accessed and `param_ix` is used to specify the expected number of elements in the array. Its value must be greater than zero and not larger than 1000. The actual size of the array is returned in the seventh word of the 8-word `id_block`. If the `param_ix` value is ≤0 or >1000, return_status equals `CM50_LCN_INDX`.

id_block—The name of a 16-byte variable where the internal ID data block is to be returned (for FORTRAN, declare as `RECORD/ID_BLOCK_STRUCT/`, or for Pascal declare as `CM50_IDBLK`).

The ID data block contents are as follows

Word 1—	Data type
Words 2..5—	Internal point identifier
Word 6—	Parameter subscript
Word 7—	Parameter qualifier (array size)
Word 8—	Enumeration set identifier

val_typ—The name of a shortword that contains a number that designates value type as follows:

1 = Real (or single element of real array)
2 = Integer (or single element of integer array)
3 = ASCII
4 = Enumeration (or single element of enumeration array)
5 = Ordinal value of enumeration (or single element of ordinal array)
6 = not used
7 = Real array
8 = Integer array
9 = Enumeration array
10 = Ordinal value of enumeration array

cg_port_num—The name of a shortword identifying the CG (1-4) to be accessed.

G.12.2 Convert LCN Time (Time Stamp)

This routine converts LCN time (a count in seconds since January 1, 1979) to external time. It accepts the LCN-time value (received as time-stamp data in response to a Get History request) and returns external time in the form MM/DD/YYΔHH:MM (where Δ represents a space). Note that the converted time is rounded down to the nearest minute.

NOTE

CONVTP is one of the few user-interface routines that is not implemented as a function. It is called as a FORTRAN subroutine or as a Pascal procedure.

G.12.2.1 Example Subroutine Call for Convert LCN Time

```
CALL   CONVTP
       (val1,
        val2,
        asctime)
```

G.12.2.2 Parameter Definitions for Convert LCN Time

val1—The name of a shortword that contains the upper-half of a value representing LCN time.

val2—The name of a shortword that contains the lower-half of a value representing LCN time.

asctime—The name of a 14-character string variable where the converted time is to be stored. (For Pascal programs, this argument is passed by %DESCR.)

NOTE

In converting LCN Time from an lcn_time_stamp_array (as returned by the get history interface calls) use the following techniques.

For FORTRAN, declare:
```
INTEGER*4 lcn_time(n)
INTEGER*2 lcn_word(2,n)
EQUIVALENCE (lcn_time, lcn_word)
```
and use `lcn_word(1,i)` for val1 and `lcn_word(2,i)` for val2

For Pascal, declare: `lcn_time : ARRAY[1..N] OF ARRAY[1..2] OF SHORTWORDS`
and use `lcn_time[i][1]` for val1 and `lcn_time[i][2]` for val2

G.13 REPLACED REL 3 USER-PROGRAM INTERFACES

G.13.1 Connect ACP to an ACIDP

This routine is called to connect an installed ACP to an ACIDP on the LCN.

G.13.1.1 Example FORTRAN Call for Connect ACP to an ACIDP

```
return_status = CM50_ACP_CON
                (%REF(acp_name),
                 %REF(acidp_name),
                 cg_port_number,
                 flags)
```

G.13.1.2 Example Pascal Call for Connect ACP to an ACIDP

```
return_status := CM50_ACP_CON
                (acp_name,
                 acidp_name,
                 cg_port_number,
                 flags);
```

G.13.1.3 Example "C" Call for Connect ACP to an ACIDP

```
return_status = cm50_acp_con
                (acp_name,
                 acidp_name,
                 &cg_port_number,
                 &flags);
```

G.13.1.4 Parameter Definitions for Connect ACP to an ACIDP

return_status—The name of an INTEGER*4 to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

acp_name—The CHARACTER*12 name of the ACP to be connected.

acidp_name—The CHARACTER*8 name of the ACIDP to connect the to the ACP.

cg_port_number—INTEGER*2 that specifies which CG (1-4) contains the `ACIDP.`

flags—This INTEGER*4 parameter sets options as described in section 9.1.3/13.1.3/17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

G.13.2 Connect DDT to ACIDP

This routine is called to connect a DDT to an ACIDP for the purpose of enabling the Data Prefetch Function in the CG. The ACIDP-ACP connection must already exist and the DDT must be CG-resident and not already connected to an ACIDP.

The `ddt_name`, and either the `acp_name`, or `acidp_name` parameters are required in the call. The Schedule, PPS and Demand parameters also are required.

G.13.2.1 Example FORTRAN Call for Connect DDT to ACIDP

```
return_status = CM50_DDT_CON
                (%REF(ddt_name),
                 %REF(acidp_name),
                 %REF(acp_name),
                 %REF(trigger),
                 flags)
```

G.13.2.2 Example Pascal Call for Connect DDT to ACIDP

```
return_status := CM50_DDT_CON
                (ddt_name,
                 acidp_name,
                 acp_name,
                 trigger,
                 flags);
```

G.13.2.3 Example "C" Call for Connect DDT to ACIDP

```
return_status = cm50_ddt_con
                (ddt_name,
                 acidp_name,
                 acp_name,
                 &trigger,
                 &flags);
```

G.13.2.4 Parameter Definitions for Connect DDT to ACIDP

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—9-character name of the DDT that is to be connected to an ACIDP.

acidp_name—8-character name of the ACIDP to which the DDT is to be connected. The `acidp_name` can be blanks if a valid `acp_name` is provided.

acp_name—12-character name of the ACP connected to the ACIDP to which the DDT is to be connected. The `acp_name` can be blanks if a valid `acidp_name` is provided

trigger—single character code with the three high-order bits assigned these meanings:

- Bit 7 : `Schedule`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 6 : `PPS` (Point_Process_Special)—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 5 : `Demand`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."

flags—This short integer sets options as described in section 9.1.3/13.1.3/17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

G.13.3 Disconnect DDT from ACIDP

This routine is called to disconnect a DDT from an ACIDP. At least one of the three parameters, `ddt_name`, `acp_name`, or `acidp_name` is required in the call (the others are passed as blanks). The ACIDP-ACP-DDT connection must already exist.

G.13.3.1 Example FORTRAN Call for Disconnect DDT from ACIDP

```
return_status = CM50_DDT_DISCON
                 (%REF(ddt_name),
                  %REF(acidp_name),
                  %REF(acp_name),
                  flags)
```

G.13.3.2 Example Pascal Call for Disconnect DDT from ACIDP

```
return_status := CM50_DDT_DISCON
                 (ddt_name,
                  acidp_name,
                  acp_name,
                  flags);
```

G.13.3.3 Example "C" Call for Disconnect DDT from ACIDP

```
return_status = cm50_ddt_discon
                 (ddt_name,
                  acidp_name,
                  acp_name,
                  &flags);
```

G.13.3.4 Parameter Definitions for Disconnect DDT from ACIDP

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—9-character name of the DDT that is to be disconnected. Can be blanks if either `acidp_name` or `acp_name` contains a valid name.

acidp_name—8-character name of the ACIDP from which the DDT is to be disconnected. Can be blanks if either `ddt_name` or `acp_name` contains a valid name.

acp_name—12-character name of the ACP connected to the ACIDP from which the DDT is to be disconnected. Can be blanks if either `ddt_name` or `acidp_name` contains a valid name.

flags—This short integer sets options as described in section 9.1.3/13.1.3/17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

G.13.4 Modify Triggers

This routine is called to modify the Triggers associated with a DDT that is connected to an ACIDP. At least one of the three parameters, `ddt_name`, `acp_name`, or `acidp_name`, is required in the call (the others are passed as blanks). The ACIDP-ACP-DDT connection must already exist.

G.13.4.1 Example FORTRAN Call for Modify Triggers

```
return_status = CM50_MOD_TRIGGERS
                (%REF(ddt_name),
                 %REF(acidp_name),
                 %REF(acp_name),
                 %REF(trigger),
                 flags)
```

G.13.4.2 Example Pascal Call for Modify Triggers

```
return_status := CM50_MOD_TRIGGERS
                 (ddt_name,
                  acidp_name,
                  acp_name,
                  trigger,
                  flags);
```

G.13.4.3 Example "C" Call for Modify Triggers

```
return_status = cm50_mod_triggers
                (ddt_name,
                 acidp_name,
                 acp_name,
                 &trigger,
                 &flags);
```

G.13.4.4 Parameter Definitions for Modify Triggers

return_status—The name of an integer to receive the overall return status of the function call. For fully successful calls, `return_status` = 1. See Appendix A.2 for an explanation and a listing of all assigned return code values.

ddt_name—9-character name of the DDT that is connected to the specified ACIDP. Can be blanks if either `acidp_name` or `acp_name` contains a valid name.

acidp_name—8-character name of the ACIDP to which the specified DDT is connected. Can be blanks if either `ddt_name` or `acp_name` contains a valid name.

acp_name—12-character name of the ACP connected to the specified ACIDP. Can be blanks if either `ddt_name` or `acidp_name` contains a valid name.

trigger—A single character code with the three high-order bits assigned these meanings:

- Bit 7 : `Schedule`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 6 : `PPS` (Point_Process_Special)—one (1) = "set prefetch on" and zero (0) = "set prefetch off."
- Bit 5 : `Demand`—one (1) = "set prefetch on" and zero (0) = "set prefetch off."

flags—This short integer sets options as described in section 9.1.3/13.1.3/17.1.3. The following flags apply to this call:

```
cm50$m_handler
cm50$m_msgon
```

SAMPLE PROGRAMS
Appendix H

This appendix describes the sample source programs supplied with the CM50S software.

The CM50$EXAMPLES directory contains the source code for a set of sample programs in FORTRAN, Pascal and "C" (and their supporting DDTs) that illustrate the use of the different groups of CM50S calls.

This appendix describes each of these sample programs. The references to points on the LCN are based on the database in the Honeywell Automation College and thus will need to be altered to run at each site.

CAUTION

The sample programs alter values to demonstrate stores to the LCN, so they should not be used with production control points in the LCN.

Remember successful stores to control parameters (SP, PV and OT) depend on the control mode of the point and the installation mode of the ACP (must be Normal) as well as the ACIDP connection.

Before running any of these programs: 1) The references to LCN points must be updated to match the installed LCN database. 2) The DDTs (if applicable) must be built. 3) The program must be compiled (using the default options for the FORTRAN, PASCAL or CC commands). and 4) If the program is to store data to the LCN, it must be installed as an ACP and attached to an ACIDP.

H.1 LINKER PROCEDURES (CM50_LNK)

The CM50_LNK.COM command file will link an ACP or DAP program to CM50S where both the object file and the executable image are in the default directory. This link procedure may be used with programs written in either FORTRAN or Pascal. Programs written in "C" must also be linked to the C runtime library, so they should use the separate CM50_LNK_C.COM command files.

Since these command procedures use the VMS default directory, the user should issue as "SET DEFAULT CM50$EXAMPLES" command before compiling and linking the sample

programs. Note that additional user object modules may be linked simply by appending their names to the argument list separated by comas (no spaces). For example:

```
@CM50$EXAMPLES:CM50_LNK testprog,myroutine
```

would create an executable image named TESTPROG.EXE in the current directory which linked both the TESTPROG.OBJ and MYROUTINE.OBJ objects to the CM50S software.

Since ACPs linked using the CM50_LNK and CM50_LNK_C procedures are not placed in the CM50$ACP directory, the full pathname of their executable files should be specified when they are installed. Example, install the sample DDTACP specifying execution path as

```
CM50$EXAMPLES.DDTACP.EXE
```

H.2 USING DDTs (DDTACP)

The DDTACP sample ACP retrieves values from the LCN, performs trivial calculations and writes the modified values back to the LCN.

It uses two DDTs, IN_SAMPLE for input, and OT_SAMPLE for output. Both of these DDTs reference the same 3 Real parameters [FC1101.SP, FC1102.SP, and CCMU20.CAPVS(1)], 1 Integer parameter [FC1101.OVERVAL], and 1 24-character ASCII parameter [CCMU20.CATAGS(1)].

Functions Invoked:
CM50_SET_ACP
CM50_DDT_GETNT
CM50_DDT_STORE
CM50_VALIDN
PRGTRM

H.3 USING MULTI-POINT LISTS (MPLACP)

The MPLACP sample ACP retrieves values from the LCN, performs trivial calculations and writes the modified values back to the LCN.

It uses specific tagnames to generate a memory structure referencing LCN points and saves that structure in a file named: MPL_SAMPLE.MPL in the user's default directory. It references the same 3 Real parameters [FC1101.SP, FC1102.SP, and CCMU20.CAPVS(1)], 1 Integer parameter [FC1101.OVERVAL] as the DDTACP example.

Functions Invoked:
CM50_SET_ACP
CM50_MPL_GENTAGS
CM50_MPL_WRITE
CM50_MPL_GET
CM50_MPL_STORE
PRGTRM

H.4 USING POINT LIST ARRAYS (PT_LIST)

The PT_LIST sample ACP converts a list of tagnames to an array of internal references, retrieves the values from the LCN, performs trivial calculations and writes the modified values back to the LCN.

It references the same 3 Real parameters [FC1101.SP, FC1102.SP, and CCMU20.CAPVS(1)] as the DDTACP example.

Functions Invoked:
CM50_SET_ACP
CM50_CONV_TAG_LIST
CM50_GET_REALNBR
CM50_STORE_REALNBR
CM50_VALIDN
PRGTRM

H.5 USING SINGLE POINT FUNCTIONS (SINGL_PT)

The SINGL_PT sample ACP retrieves a value from the LCN, performs trivial calculations and writes the modified value back to the LCN.

It references two Real parameters [FC1101.SP and CCMU20.CAPVS(1)].

Functions Invoked:
CM50_SET_ACP
CM50_CONV_TAG
CM50_GETPT_ID
CM50_GET_ID
CM50_STORE_ID
PRGTRM

H.6 ACCESSING HISTORY (HISTORY)

The HISTORY sample program retrieves values from the History Module LCN. It does not have to be installed as an ACP, but the tags it references must be configured into a history group on the LCN.

It uses one DDT, H_SAMPLE for history input, which references two Real parameters [FC1101.SP and FC1102.SP]. FC1101.SP is also used directly for a single-point retrieval., 1 Integer parameter [

Functions Invoked:
CM50_SET_ACP
CM50_DDTHIS_SNAP
CM50_DDTHIS_AVERT
CM50_CONV_PT
CM50_PTHIS_AVER
CM50_VALIDN
CM50_TIMLCN_VAXA
CM50_TIMVAXA_ASC
PRGTRM

H.7 MANAGING ACPs (ACP_ADMIN)

The ACP_ADMIN sample program retrieves a list of installed ACPs, alters the installation mode of the DDTACP sample ACP to "TEST", activates it and returns it back to "NORMAL" mode.

Functions Invoked: CM50_ACP_LISTALL
CM50_ACP_CHG_MODE
CM50_ACP_ACT
CM50_ACPDELAY

H.8 MANAGING DDTs (DDT_ADMIN)

The DDT_ADMIN sample program retrieves a list of installed DDTs and the detailed information about one of them, then rebuilds the fifth DDT, maintaining any existing prefetch ACIDP connection.

Functions Invoked: CM50_DDT_LIST
CM50_DDT_DETAIL
CM50_DDT_DISCON
CM50_DDT_UNINST
CM50_DDT_BUILD
CM50_DDT_CON

H.9 READING THE CG DATABASE (CG_BASE)

The CG_BASE sample program retrieves lists of the ACIDPs, CRDPs and DDTs that are resident in CG 1.

Functions Invoked: CM50_CG_RDDT
CM50_CG_CRDP
CM50_CG_ACIDP

H.10 USING LCN FILE TRANSFER FUNCTIONS (LCN_XFER)

The LCN_XFER sample program lists files and directories on a history module and retrieves a file from the LCN.

It references two site-specific variables: HM_NODE (initially set to '49', should be modified to match the node number on the local LCN), and FILE_PATH (should be modified to match an ASCII file on the local LCN). The program will create two files in the VAX: LCN_XFER.ASC (an ASCII file) and LCN_XFER_ASC.LA (its associated Attributes file).

Functions Invoked: CM50_FILE_LIST
CM50_HM_LISTF
CM50_ATTR_LIST
CM50_LCN_READ

READER COMMENTS

Honeywell IAC Automation College welcomes your comments and suggestions to improve future editions of this and other publications.

You can communicate your thoughts to us by fax, mail, or toll-free telephone call. We would like to acknowledge your comments; please include your complet name and address

BY FAX: Use this form; and fax to us at (602) 313-4108

BY TELEPHONE: In the U.S.A. use our toll-free number 1*800-822-7673 (available in the 48 contiguous states except Arizona; in Arizona dial 1-602-313-5558).

BY MAIL: Use this form; detach, fold, tape closed, and mail to us.

Title of Publication: **CM50S User Manual** Issue Date: **7/93**

Publication Number: **CM11-430**

Writer: **Terry Rippstein**

COMMENTS: _____

_____

_____

_____

_____

_____

_____

_____

RECOMMENDATIONS:_____

_____

_____

_____

_____

_____

_____

_____

NAME _____ DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

TELEPHONE _____ FAX _____

(If returning by mail, please tape closed; Postal regulations prohibit use of staples.)

Communications concerning technical publications should be directed to:

Automation College
Industrial Automation and Control
Honeywell Inc.
2820 West Kelton Lane
Phoenix, Arizona 85023-3028

FOLD FOLD

From: _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE USA

BUSINESS REPLY MAIL

FIRST CLASS **PERMIT NO. 4332** PHOENIX, ARIZONA

POSTAGE WILL BE PAID BY ....

Honeywell

Industrial Automation and Control
2820 West Kelton Lane
Phoenix, Arizona 85023-3028

Attention: Manager, Quality

FOLD FOLD

Cut Along Line

Additional Comments:

Honeywell

Industrial Automation and Control
Honeywell Inc.
16404 North Black Canyon Highway
Phoenix, Arizona 85023-3033

Helping You Control Your World