Created:January 21, 1996
Modified:March 6, 2000

Gemini Data Handling System Report

ICD 1c — Baseline DHS Interface

Norman Hill, Séverin Gaudet, Dayle Kotturi

dhs_pdr_icd1c/07

This document describes the baseline low level command interface to the DHS.

1.0 Introduction

The DHS is a UNIX-based principal system of the Gemini control system. The DHS is not an EPICS-based system, and so cannot use the interface defined in [6]. This document describes an interface based on the IMP message passing system [15] which provides the functionality required by OCS and other Gemini principal systems described in [5].

1.1 Purpose

This document describes the low-level command interface to the DHS. The baseline Gemini principal systems interface is described in [5]. This interface will be used by the OCS to send any and all commands to the DHS and its components. It also may used by other Gemini principal systems to send commands to the DHS, and will form the basis of the bulk data transfer interface described in [8].

In detail this document:

1. Indicates the nature and behaviour of the software interface.
2. Describes programmatic interface.
3. Provides examples of the use of this interface.

1.2 Scope

This document defines the low level command interface to be used by the DHS and all DHS supplied components. This document does not list the details of the commands and/or attributes which will be carried by this interface. The DHS commands, their attributes and responses will be covered in detail in [10].

1.3 References

[1] *Gemini Software Design Description*, SPE-C-G0037, Gemini 8m Telescopes Project

[2] *Gemini Software Programming Standards,* SPE-C-G0009/02, Peregrine M.McGehee, Steve Wampler

[3] *Gemini System Interfaces*, GSCG.grp.005, Gemini 8m Telescopes Project

[4] *Overview of System Interfaces*, GSCG.grp.006, Gemini 8m Telescopes Project

[5] *ICD 1a — The System Command Interface*, GSCG.KKG.009/009, Kim Gillies, Gemini 8m Telescopes Project

[6] *ICD 1b - The Baseline Attribute/Value Interface*, GSCG.grp.024, Kim Gillies, Steve Wampler, Bret Goodrich, Gemini 8m Telescopes Project

[7] *ICD 2 — Systems Status and Alarm Interface*, Version GSCG.grp.013-ICD2/005, Kim Gillies, Gemini 8m Telescopes Project

[8] *ICD 3 — Bulk Data Transfer*, icd3/23, Norman Hill and Séverin Gaudet HIA

[9] *ICD 4 —Logging Information*, dhs_pdr_icd4/10, Norman Hill and Séverin Gaudet HIA

[10] *ICD 3.2 — The DHS Command Interface*, dhs_icd3.2/05, Norman Hill, Dayle Kotturi and Séverin Gaudet, HIA

[11] *Baseline Major Systems Interface*, GSCG.kkg.005, Version 003, 28/6/94, Gemini 8m Telescopes Project

[12] *OCS Behavioural Interface Model Report*, GSCG.kkg.002, Version 001, 23/6/94, Gemini 8m Telescopes Project

[13] *Report on the Interface Model Issues*, GSCG.grp.001, Gemini 8m Telescopes Project

[14] *Virtual Telescope Interface,* Jeremy Bailey, UKIRT System Note 6.3, 8 November, 1991

[15] *Interprocess Message Passing (IMP) System*, AAO/IMP_MANUAL_8, DRAMA Software Report 8, Version 1.3, 13 June 1997, Keith Shortridge, Anglo-Australian Observatory

[16] *Self-defining Data System (SDS)*, AAO/SDS_07, Jeremy Bailey, Anglo-Australian Observatory, Version 1.2, 9 September, 1992

[17] *Glossary for the Gemini Software*, GSCG.grp.016, Gemini 8m Telescopes Project

1.4 Abbreviations and Acronyms

AAO	Anglo-Australian Observatory.
AV	Attribute/value.
DHS	Data Handling System.
DRAMA	Distributed Real-time AAO Monitor for Astronomy.
EPICS	Experimental Physics and Industrial Control System.
GCS	Gemini Control System.
GSCG	Gemini Software and Controls Group.
ICD	Interface Control Document.
ICS	Instrument Control System.
IMP	Interprocess Message Passing System.
LAN	Local Area Network.
N/A	Not Applicable.
OCS	Observatory Control System.

PDF	Parameter Description File.
PSBI	principal Systems Baseline Interface.
SDD	Software Design Description.
SDS	Self-defining Data System.
TCP/IP	Transmission Control Protocol/Internet Protocol.
TCS	Telescope Control System.
VME	A real-time system obeying the ANSI/IEEE 1014-1987 Versatile Backplane Bus standard.

1.5 Glossary

Additional glossary information is available in [17].

Action — When a system is sent a command, it responds with some behavior. This behavior is the *action* associated with that command. Acceptance of the command by the system results in the initiation of the action. A single action may be composed of multiple subsystem actions.

Action Completion — An action that is initiated by some command is considered complete when the target system has either achieved the state described by the command, or is incapable of doing so for some reason (typically an error). Action completion is distinct from any status information that might also be associated with the mechanisms performing the action.

Attribute — An attribute is a textual description of some part of a Gemini based hardware or software system. An attribute has an associated value.

Attribute Value — The value is the data associated with a particular attribute.

Attribute Value List — A list of zero or more attributes and their associated values.

Client system — A Gemini principal system using this interface to send commands or data to the DHS and to receive any resulting responses.

Command — A specific request for an action on the part of the server. A command may have a list of attributes and values indicating the details of the requested action (also known as the configuration). A command may produce a list of attributes and values as a command response. A command has a current status which may change during the execution of the command.

Command Completion — Commands are considered as ‘complete’ after they have been recognized and accepted or rejected by a principal system. This is analogous to the concept of command completion in the Virtual Telescope model [14], where actions are assumed to be instantaneous.

Command Response — One or more results generated by a server in response to a command. A command response is in the form of an attribute value list.

Command Server — Subsystem of the DHS responsible for receiving and executing commands.

Command Status — The current status of a command. The command status will change as the execution of the command proceeds.

Command Tag — A value identifying a specific instance of a command.

Configuration — A set of attributes and their associated values describing the differences needed to move the system from one state to another state.

DHS library — A library supplied by the DHS work package containing the public programmatic interface to the DHS system.

Parameter Description File — A parameter description file describes the commands and their associated attributes accepted by a system.

Principal System — At the highest level in the GCS software decomposition, the software system is divided into four kinds of software systems called principal systems. The four types are called: the Data Handling System, the Observatory Control System, the Telescope Control System, and the Instrument Control System. There may be up to four concurrently executing Instrument Control Systems.

Server system — System responsible for responding to commands sent over this interface. The server system will be a part of the DHS system.

String — Null terminated "C" style string of ASCII characters.

System Configuration — The portion of a configuration that is unique to a single system is called a system configuration.

Systems Interface — The principal systems within the GCS software design interact with one another through the Systems Interface.

UNIX — A standard host-level operating system. The Gemini baseline for "UNIX" is "Solaris" marketed by Sun Microsystems.

VxWorks — A real-time operating system for VME hardware marketed by Wind River Systems.

1.6 Stylistic Conventions

1.6.1 References to other documents.

References to other documents are given using a number in square brackets, for example [1]. The exact references are given in Section 1.3 on page 2.

1.6.2 References to the Gemini systems

The term "ICS" is used in statements which refer equally well to any Instrument Control System. Other Gemini systems are referenced by their acronyms as they appear in Section 1.4 on page 2.

1.6.3 Functions and parameters

Function and parameter names are given in mixed case as specified in [2]. Data types are written in upper case. Functions are described by showing their ANSI C prototype definitions. Examples code conforms to the guidelines given in [2].

1.6.4 Typographic conventions

Program text is shown in the `Courier` type face. References to symbols in the DHS library are shown in **bold** type face.

2.0 Overview

The Gemini Control System (GCS) is made up of principal software systems: the Telescope Control System (TCS), the Data Handling System (DHS), the Observatory Control System (OCS), and one or more Instrument Control Systems (ICS). The purpose of these systems, and the way they interact with each other, is described in [1].

ICD 1c describes a low level mechanism by which any system in the GCS can send commands to the DHS.

2.1 System Hardware Architecture

The overall layout of the Gemini computer hardware and communication equipment is shown in Figure 1 on page 6 which is taken from [3]. The interface described in this document will be able to communicate with the DHS over any of the LANs shown in Figure 1 provided they are able to carry TCP/IP traffic, however in general, ICD 3 bulk data transfer traffic should be carried by the data LAN, and commands, and responses should be done through the control LAN. The LAN used will be determined by the client system when it opens a connection to the DHS.

2.1.1 The Data Handling System (DHS)

The DHS is assumed to be based on a UNIX work station (baselined as a Sun Sparcstation running Solaris 2.X). No other DHS architecture is currently supported.

2.1.2 The Observatory Control System (OCS)

The OCS is assumed to be based on a UNIX work station (baselined as a Sun Sparcstation running Solaris 2.X). No other OCS architecture is currently supported.

2.2 Context Diagram

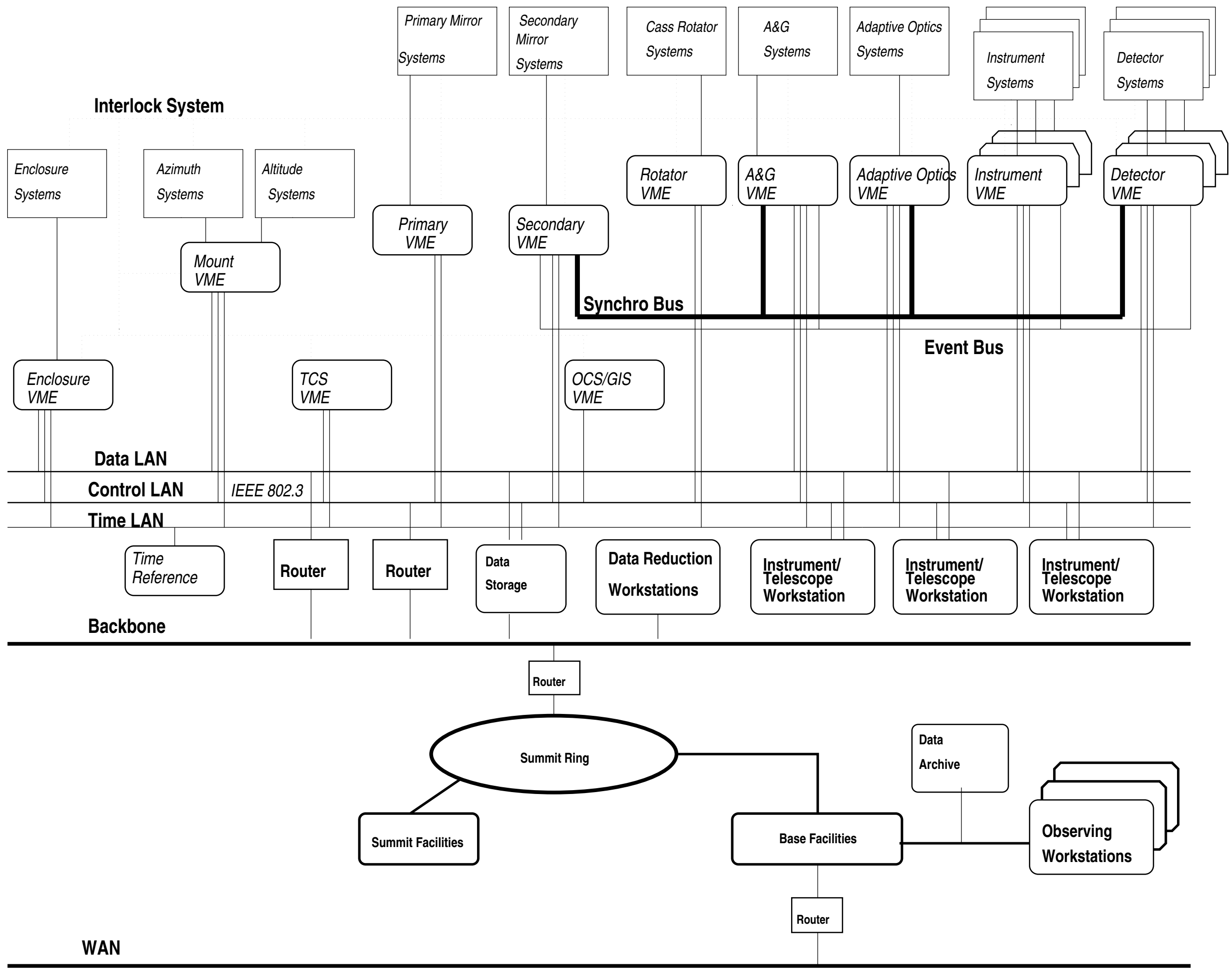
Figure 2 on page 7 shows a context diagram of the Gemini Software System as seen from the point of view of the DHS command interface. The Gemini systems appear to the interface as entities in its outside world. The bulk data transport data flows shown in Figure 2 and labelled as ICD 3 data flows are described in detail in [8]. The details of the commands, configuration attributes, and command response attributes sent via the ICD 1c interface will be covered in the PDF document [10].

3.0 Behaviour

The DHS work package provides a C library which can be linked with UNIX or VxWorks programs to allow Gemini Principal systems to communicate with the DHS and its components. This library will be referred to as the DHS library in the rest of this document. The DHS library allows clients to establish connections to command servers, create attribute value lists, issue commands to command servers, and to monitor the responses to commands.

There are two fundamentally different modes in which the ICD 1c interface can be used. The primary mode involves setting application supplied callback functions which the DHS library will call when various events occur. The callback functions are responsible for correctly dealing with the events. In this mode, the application would start a DHS library event loop which then runs continuously, executing callback functions as required. An option is provided to allow the application to run the event loop in a thread, freeing the application to perform other tasks.

FIGURE 1. The System Hardware Architecture

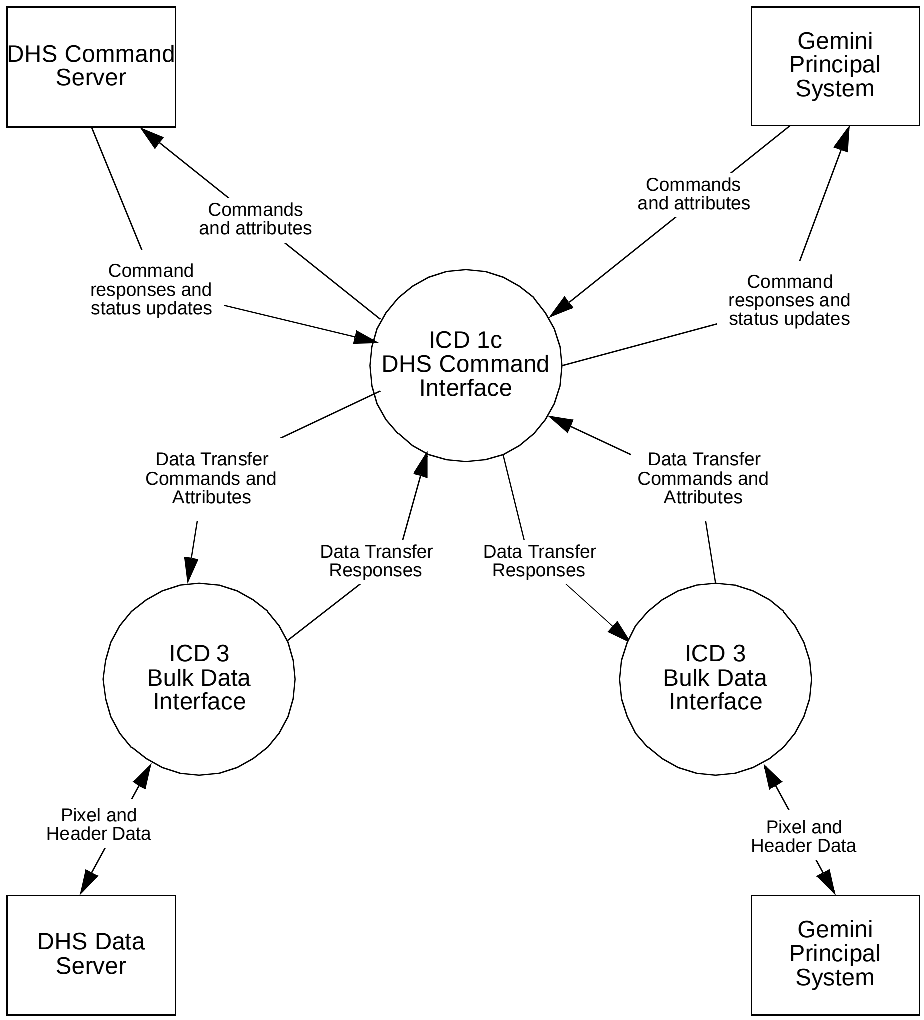


In the second mode of the ICD 1c interface, the client systems issue commands and then either poll the status of the commands or wait for commands to reach a terminal state and then deal with the responses to the commands. Using this mode, the DHS library will run the event loop whenever it is required.

The interface to the DHS library is the same for both modes. The two modes are not mutually exclusive, so it is possible for an application to wait for commands and query command status and command responses even if callbacks functions are being used, and any supplied callback functions will be executed when doing polling. (I'm not sure why anyone would want to mix modes since the same information available by polling is available in the callback function parameters, but there is no reason to prevent it.)

The details of the public interface to this library are described in Section 5.0 on page 11. The required behaviour of the interface is described in [5].

FIGURE 2. Context Diagram

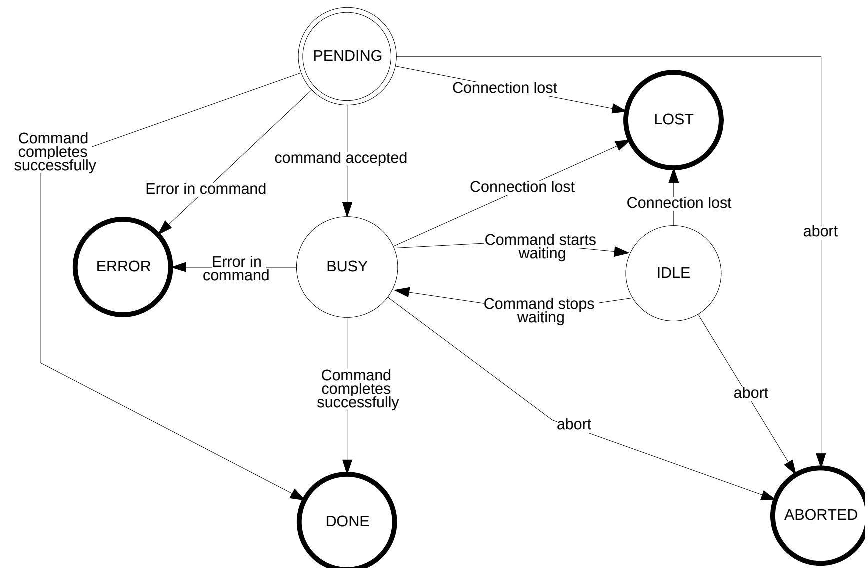


The ICD 1c interface works by allowing a command client to establish a connection to a command server, and send commands over the resulting connection. All commands sent on the ICD 1c interface consist of a command name and an optional attribute value list containing the commands input parameters. A command may produce one or more optional responses which also consist of attribute value lists. After a command has been accepted by a server system, the DHS library function **dhsApply** which issues commands, returns.

The commands issued on the ICD 1c interface have an associated state. The state transition diagram for commands is shown in Figure 3 on page 8 (each of the states in the diagram should be prefixed with “**DHS_CS_**” to form the symbolic value used by the DHS library). The command states are described in Section 6.1.6 on page 16. The DHS library function **dhsApply** returns when the command leaves the **DHS_CS_PENDING** state. This is the command completion state as defined by ICD 1a [5]. All other state changes are sent to the client asynchronously as part of a command response. Most commands will take the **DHS_CS_PENDING** →

DHS_CS_BUSY → **DHS_CS_DONE**, or **DHS_CS_PENDING** → **DHS_CS_DONE** paths through the state diagram. The action completion state is achieved whenever the command state is one of the end state values shown in Figure 3 on page 8.

FIGURE 3. Command state transition diagram



a. All states should be prefixed with "**DHS_CS_**" to form the symbolic value used by the DHS library.

The DHS library provides functions to allow clients to establish connections to command servers, create attribute value lists, issue commands to command servers, and to monitor the responses to commands. All functions of the DHS library are thread safe.

3.1 Events and Responses

The significant events affecting the ICD 1c interface and their correct responses are listed in Table 1 on page 8. All events are initiated by the client system, except the initial registering of a server system and abnormal termination of client or server systems.

TABLE 1. Events and Responses

Events	Response
Starts and registers its name with the **dhsInit** function	The name of the server is registered with the IMP system, and the server becomes available for client connections.
Client connects to server	Server accepts connection.
Client disconnects from server	Server drops connection.
Attributes are set in an attribute value list.	The attribute is added to the attribute value list which remains stored on the client system.

TABLE 1. Events and Responses (Continued)

Events	Response
Command and attribute value list are sent to the server	The command name and attribute list are sent to the server system. The command callback function on the server system evaluates the command and attributes to ensure they are valid and accepts or rejects the command. If the command is rejected, a message will be returned indicating the reason for rejection, and the command state is set to **DHS_CS_ERROR**. The function call to send the command will not return until the command has been accepted or rejected. Command status of an accepted command would typically be set to **DHS_CS_BUSY** initially, however it is possible for a command action to complete immediately, in which case the initial command status would be **DHS_CS_DONE**. If it has been set, the response callback function will be called to set the initial state of the command.
Command is stopped	The server system stops the action at the next reasonable stop point (this will be determined during the design of the server system). The command status would be set to **DHS_CS_ABORTED** if the command is stopped before normal completion. Not all commands can be stopped, e.g. it is probably not practical to allow a "reset" command to be aborted since the system may be left in an inconsistent state. If the command is stopped, the client's response callback function will be executed to indicate the change in status of the command.
A command produces a response	The status information and/or command response is sent from the server system to the client system. The ICD1c library updates the current status of the command and adds any response to the accumulated responses for the command. If a response callback function has been set, the function will be called, otherwise the client system is responsible for polling the active commands to detect the response.
An action completes	The server system updates the status of the command to **DHS_CS_DONE**. If it has been set, the response callback function is called. The client should free the command tag at this point.
Error occurs in the DHS library event loop	Client supplied error callback function is executed.
Client system queries command status	Current command status is returned.
Command server crashes	The DHS library detects the loss of connection. The status of any commands active on the connection are set to **DHS_CS_LOST** by the DHS library, and from the point of view of the client, the command action is complete the command tag should be freed. The client automatically attempts to reconnect to the command server for the period of time specified by the command server. Since the command server is not restarted, the attempts to reconnect will fail. The DHS library will execute the error callback function with the **DHS_E_CON_LOST** status.

TABLE 1. Events and Responses (Continued)

Events	Response
Command server disconnects and then becomes available (i.e. is reset).	The DHS library detects the loss of connection. The state of all commands currently active on the connection are set to **DHS_CS_LOST**. The client automatically attempts to reconnect to the command server for the period of time specified by the command server. The connection callback routine is executed on the client, with the connection state set to **DHS_CON_RECONNECT**. The connection state is set to **DHS_CON_CONNECTED** after the connection callback returns. It is the responsibility of the client application to clean up any incomplete commands sent to the server. It is the responsibility of the server to avoid unnecessary disconnects while commands are incomplete.
Client Crashes	Any attempt by the server to send a response to the client results in an error. The connection to the client is terminated. The server may complete active command or abort the commands, at the discretion of the server.

4.0 Implementation Overview

The DHS library will be a C library based on the DRAMA IMP messages passing system [15], and DRAMA SDS data structures[16]. The underlying messages passed using IMP are listed in Table 2 on page 10. The only provision for “command control” functions like “abort command” is the ability to pass a command tag as an argument to a server. This ability will allow the “command control” functionality to be defined in the PDF files for the DHS. Any security requirements can be met if the command server refuses all commands except an “authorize” command until the authorize command has been issued with acceptable attributes.

TABLE 2. IMP messages used by ICD 1c

Message	Description	Parameter	Type
Command	Contains information describing a command.	Command name	String
		Command tag	Integer
		Connection id	Integer
		AV list	SDS
Command reply	Contains a response to a command.	Command tag	Integer
		Connection id	Integer
		Response AV list	SDS
		Command status	Enum
		Status description	String
Connection accept	Accepts a connection.	Connection id	Integer
Connection ok	Send in response to a connection verify message.	Command tag	Integer
Connection refuse	A server refuses a connection.	Connection id	Integer

TABLE 2. IMP messages used by ICD 1c (Continued)

Message	Description	Parameter	Type
Connection Verify	Request that a system verify the connection is usable.	none	
Disconnect	Indicates a client is disconnecting.	Connection id	Integer
Command reply	Contains a response to a command.	Command tag	Integer
		Connection id	Integer
		Response AV list	SDS
		Command status	Enum
		Status description	String
Server Exit	Request that a client close its connection.	none	

The underlying structures used by the DHS library are SDS structures. This allows flexible use of attributes and their values, and automatically allows for any differences in data storage format between different systems.

The DHS library is thread safe, permitting its free use in threaded applications.

5.0 The Programmatic Interface

All public symbols in the DHS library are prefixed by the string "dhs" or "DHS". Typical uses of the ICD 1c interface are described in Section 5.1 on page 11 and Section 5.2 on page 12 and example code is given in Section 7.0 on page 33. Although the interface is described in terms of "client" use of the library and "server" use of the library, the interface is basically bidirectional, and it is possible for "servers" to send commands to "clients" if the appropriate callback routines have been set up by the client. The only clear distinction between client and server is that the client is the one that uses the **dhsConnect** function call to establish the initial connection.

It is acceptable for a system that accepts connections to also call the **dhsConnect** function to connect to other servers. In this case the program is acting as both a client and server at different times.

5.1 Client use of the interface

Typically, a client using the ICD 1c interface would:

1. Initialize the library with **dhsInit**.
2. Set any callback functions with the **dhsCallbackSet** function. The error callback must be set at a minimum, and most clients will set callbacks for command responses.
3. Make one or more connections to DHS servers with the **dhsConnect** function.
4. Start the event loop in a thread with the **dhsEventLoop** function (optional). If no event loop is started, the DHS library will run the event loop when it has control and is waiting for events.
5. Use the **dhsAvListNew** and **dhsAvAdd** functions to create and populate an attribute value list.

6. Use the **dhsApply** function to send messages to the DHS.
7. Continue processing and allow the response callback function (see Section 6.6.2 on page 32) to monitor command status changes and responses, or use the **dhsWait** and **dhsStatus** functions to monitor the progress of commands, depending on the use of the library.
8. In a callback routine, or after polling to ensure the command action is complete, the client would examine responses with the attribute parsing functions **dhsResponseGet** and **dhsAvInfo.** If necessary the client callback routine may create a new thread in order to process the response. Creating threads in client callback routines may be necessary since the event loop will not process new events until the client's response callback routine returns.
9. Close connections with the **dhsDisconnect** function (optional).
10. Stop the event loop with the **dhsEventLoopEnd** function (optional).
11. Clean up the library with the **dhsExit** function.

Generally steps 1 through 4 are initialization, and steps 9 through 11 are cleanup, although the library may be initialized and exited repeatedly, callback functions may be set or changed at any time, connections may be opened and closed as required, and the event loop may be started and stopped at any time. If steps 9 and 10 are omitted, they will be performed automatically by the **dhsExit** function.

A user data pointer may be provided as an parameter to the **dhsApply** or **dhsConnect** functions, or set with the **dhsConUserDataSet** and **dhsUserDataSet** functions. (Note that the UserDataSet functions should be used with extreme caution, it is always possible for events to occur before the user data pointer has been set.) If provided by the application, the user data pointer will be associated with the command tag or connection. User data is not passed to the server or interpreted by the DHS library in any way, but is retrievable with the **dhsUserDataGet** or **dhsConUserDataGet** functions and the command tag user data pointer will be passed as an argument to the command response callback function. The purpose of the user data pointer is to allow an application to supply command or connection specific data which is easily available in the command response callback function. The user data will probably consist of a pointer to an application supplied C structure, C++ object, or function (C or C++), but the use of this pointer is entirely up to the application programmer and may be anything that may be referenced by a void pointer. The user data pointer provides opportunities for interesting bugs associated with this style of programming (passing pointers to data that may or may not be valid when the callback is executed, etc.) and should be used with caution.

5.2 Server use of the interface

Typically, a server using the library would:

1. Initialize the library with the **dhsInit** function.
2. Set any callback functions with the **dhsCallbackSet** function. The error callback and the command callback functions should both be set.
3. Probably start the event loop with the **dhsEventLoop** function, but not in a separate thread. Most servers will not be required to perform any tasks between servicing commands so it would not be necessary to have a separate thread for the event loop.
4. When the command callback function is called the server will evaluate the command and attribute value list and set the appropriate initial command state with the **dhsCmdResponse** function. The server will then perform the actions required for the command, returning any responses or status updates with the **dhsCmdResponse** function. Multiple responses may be sent by calling the **dhsCmdResponse** function more than once. The last call to the **dhsCm-**

dResponse function must set the command status to a terminal state (**DHS_CS_ABORTED**, **DHS_CS_DONE**, or **DHS_CS_ERROR**).

Note that until the command callback routine returns, the DHS library event loop will be unable to process new events. If necessary, a server can create new threads to handle the processing of commands to allow the command callback function to return, and event processing to continue.

5. When some exit condition is met (possibly a command received through the interface, or a UNIX signal) the **dhsEventLoopEnd** function should be used to stop the event loop.
6. Clean up the library with the **dhsExit** function.

Steps 1 through 3 are initialization, and steps 5 and 6 are cleanup.

5.3 Interface summary

The data types used by the ICD 1c interface are summarized in Table 3 on page 13 and the ICD 1c interface functions are summarized in Table 4 on page 14. The reference column in the tables indicates the section of this document where a detailed description of the item will be found.

TABLE 3. ICD 1c data type summary

Data Type	Description	Reference
DHS_AV_ID	An attribute in an attribute value list	6.1.1
DHS_AV_LIST	An attribute value list	6.1.2
DHS_BOOLEAN	Contains a boolean value	6.1.3
DHS_CB_FN_PTR	Pointer to a callback function	6.1.4
DHS_CB_TYPE	Type of callback function	6.1.5
DHS_CMD_STATUS	Contains the current status of a command	6.1.6
DHS_CON_STATE	The state of a connection	6.1.7
DHS_COND_FN	Function used to test a condition in an event loop	6.1.8
DHS_CONNECT	Identifies a client or server connection	6.1.9
DHS_DATA_TYPE	Type of a data item	6.1.10
DHS_DEBUG_LEVEL	Debugging level of the DHS library	6.1.11
DHS_EL_TYPE	DHS library event loop type	6.1.12
DHS_ERR_LEVEL	Error level of an error status	6.1.13
DHS_STATUS	Status value returned by functions in the DHS library	6.1.14
DHS_TAG	Identifies a specific command	6.1.15
DHS_THREAD	Identifies a thread.	6.1.16

6.0 Detailed Interface Description

The following sections detail the ICD 1c public interface. The function descriptions are placed in sub-sections, based on the situations where a function would normally be used, for example all functions which would normally be only used by a client system are described in Section 6.3 on

TABLE 4. ICD 1c interface function summary

Function	Description	Reference
General functions		
dhsInit	Initialize the DHS library	6.2.1
dhsExit	Clean up the DHS library	6.2.2
dhsMessage	Get the DHS library error numbers and level, and a pointer to the message string	6.2.3
dhsMessageClear	Clear a DHS message	6.2.4
dhsEventLoop	Start the DHS event loop	6.2.5
dhsEventLoopEnd	Stop the event loop	6.2.6
dhsCallbackSet	Set a DHS library callback function	6.2.7
dhsDisconnect	Disconnect from a server	6.2.8
dhsDebugLevel	Set the debugging level of the DHS library	6.2.9
dhsElBroadcast	Cause the event loop to execute on no-op iteration	6.2.10
dhsIsConnected	Check a connection to see if it is still connected	6.2.11
Client functions		
dhsConnect	Connect to a server	6.3.1
dhsConUserDataGet	Get a connection's user data pointer	6.3.2
dhsConUserDataSet	Set a connection's user data pointer	6.3.3
dhsConnectInfo	Returns information about the other end of a connection	6.3.4
dhsApply	Sends a command to a server	6.3.5
dhsWait	Wait for a list of command actions to complete	6.3.6
dhsStatus	Query the current status of a command	6.3.7
dhsTagFree	Free a command tag	6.3.8
dhsUserDataGet	Get the user data pointer for a command	6.3.9
dhsUserDataSet	Set the user data pointer for a command	6.3.10
dhsResponseGet	Get the response associated with a command	6.3.11
dhsTagDone	Check to see if a command tag is in one of the end states	6.3.12
Server functions		
dhsCmdResponse	Send a response to a command	6.4.1
dhsServerExit	Request all clients close their connections	6.4.2
dhsSetTimeout	Set the amount of time a server expects to take to restart	6.4.3
Attribute value list manipulation functions		
dhsAvListNew	Allocate a new attribute value list	6.5.1
dhsAvListFree	Free an attribute value list	6.5.2
dhsAvAdd	Add an attribute and its value to an AV list	6.5.3
dhsAvDelete	Delete an attribute and its value from an AV list	6.5.4

TABLE 4. ICD 1c interface function summary (Continued)

Function	Description	Reference
dhsAvInfo	Get information about an attribute	6.5.5
dhsAvFind	Find an attribute by name	6.5.6
dhsAvIndex	Find an attribute based on its position in the AV list	6.5.7
dhsAvListAccess	Access an AV list in a memory buffer	6.5.8
dhsAvListCopy	Copy an attribute value list	6.5.9
dhsAvListExport	Export an attribute value list to a memory buffer	6.5.10
dhsAvListPrint	Print an AV list to standard out	6.5.11
dhsAvListSize	Return the size of an AV list	6.5.12
User supplied functions		
error callback	User supplied error callback function	6.6.1
response callback	User supplied command response callback function	6.6.2
command callback	User supplied command callback function	6.6.3
connection callback	User supplied connection callback function	6.6.4

page 24. It is assumed that the header file “dhs.h” has been included before any of the DHS library functions are used. All DHS library functions have an output parameter of type **DHS_STATUS** as the last parameter, which returns the completion status of the function. After successful completion a function will return the value **DHS_S_SUCCESS**. If any function is called with the status value not equal to **DHS_S_SUCCESS**, the function will return immediately without modifying the status value. (Note: one of the more common errors encountered while programming with the DHS library is not initializing a status variable to DHS_S_SUCCESS.) The return status values are defined in dhs.h and listed in Table 5 on page 18.

6.1 Data types

The following data types are defined in file **dhs.h**, and used by the ICD 1c interface:

6.1.1 DHS_AV_ID An attribute in an attribute value list

This data type is used to identify an individual attribute in an attribute value list. The attribute is returned by functions **dhsAvFind** and **dhsAvIndex**, and is passed as an argument to function **dhsAvInfo**.

6.1.2 DHS_AV_LIST An attribute value list

This data type is used by an application to contain an attribute value list to be passed to a command. This data type is also used to contain the attribute value list sent as a response to a command. The attribute value list can be manipulated by the DHS library functions **dhsAvListNew**, **dhsAvListFree**, **dhsAvAdd**, **dhsAvInfo**, and **dhsAvFind**.

6.1.3 DHS_BOOLEAN Contains a boolean value

This is an enumerated data type whose value can be **DHS_TRUE** (1) or **DHS_FALSE** (0).

6.1.4 DHS_CB_FN_PTR — Pointer to a callback function

This is a pointer to a function that returns nothing and has an undetermined parameter list. This is used to point to any callback function.

6.1.5 DHS_CB_TYPE — Type of callback function

This is an enumerated type used to indicate the type of callback function being set in function **dhsCallbackSet**. Possible values of this type are:

DHS_CBT_ERROR — DHS library error callback function.

DHS_CBT_RESPONSE — Callback function for command responses.

DHS_CBT_COMMAND — Callback function for command reception.

DHS_CBT_CONNECT — Callback function for connection state changes.

6.1.6 DHS_CMD_STATUS — Contains the current status of a command

This is an enumerated data type which will contain the current status of a command. Possible values of this data type are:

DHS_CS_IDLE — Command is waiting to execute, or waiting for some event before it proceeds.

DHS_CS_BUSY — Command is in the process of executing.

DHS_CS_ERROR — Command action has completed with an error.

DHS_CS_ABORTED — Command was aborted by another command.

DHS_CS_DONE — Command action has completed successfully and all results have been reported.

DHS_CS_PENDING — Command is being evaluated by the command server.

DHS_CS_LOST — The connection to the server was lost before the command reached a terminal state. Unlike the other command statuses, this status is set by the client side of the DHS library.

6.1.7 DHS_CON_STATE — The state of a connection

A value of this type is passed as the second argument to the connection callback function to indicate the current state of the connection. Possible values for this type are:

DHS_CON_CONNECTED — The connection is open and ready to be used.

DHS_CON_DISCONNECTING — The connection is in the process of disconnecting.

DHS_CON_DUPLICATE — The connection is a duplicate of an already existing connection. (This should only be visible within the DHS library.)

DHS_CON_ERROR — The connection is in an error state and should not be used.

DHS_CON_LOCATED — A requested server has been located, but the connection is not yet complete. (This should only be visible within the DHS library.)

DHS_CON_LOST — The server has exited without properly closing the connection.

DHS_CON_NOT_ACCEPTED — The connection is open but the client has not yet determined if it will accept the connection or not. (This should only be visible within the DHS library.)

DHS_CON_NOT_CONNECTED — The client is attempting to locate the requested server. (This should only be visible within the DHS library.)

DHS_CON_RECONNECT — A client has automatically reconnected to a server. This is a transient state, the state of the connection should change to **DHS_CON_CONNECTED** after the connection callback function is executed.

DHS_CON_REFUSED — The server has refused the connection from the client.

DHS_CON_REQUESTED — This state only exists on a server for a connection requested by a client, but which the server has not yet accepted or rejected.

DHS_CON_TIMEOUT — Not used.

6.1.8 DHS_COND_FN — Function used to test a condition in an event loop

This is a pointer to a function that takes a void pointer as an parameter and returns a boolean value. This data type is only used as an parameter to function **dhsEventLoop** when the *loopType* parameter is **DHS_ELT_COND**.

6.1.9 DHS_CONNECT — Identifies a client or server connection

This value is used to identify a connection to a client or server.

6.1.10 DHS_DATA_TYPE — Type of a data item

This is an enumerated data type which indicates the type of a data item. Valid values for this data type listed in Table 6 on page 33. The **DHS_DT_TAG** type allows commands to reference previous command, allowing things like “abort command” commands. The **DHS_DT_AVLIST** data type allows nested attribute value lists, although this is only expected to be used by the ICD 3 bulk data transfer interface.

6.1.11 DHS_DEBUG_LEVEL — Debugging level of the DHS library

This is an enumerated type indicating the amount of information the DHS library will return during its processing. Possible values of this type are:

DHS_DEBUG_OFF — No additional debugging information are reported.

DHS_DEBUG_ON — A moderate amount of debugging information are reported.

DHS_DEBUG_FULL — Large amounts of debugging information are reported.

6.1.12 DHS_EL_TYPE — DHS library event loop type

This is an enumerated type indicating how the DHS event loop should be run. See the description of function **dhsEventLoop** for a description of the use of this data type. Possible values of this type are:

DHS_ELT_THREADED — The DHS library should create a thread for the event loop and then return.

DHS_ELT_WAIT — Start the event loop in the current thread.

DHS_ELT_COND — Run the event loop until some condition is met. This option is required internally for the DHS library and may not be of any use to applications.

DHS_ELT_NO_WAIT — Process all pending events and then return. This option is required internally for the DHS library and may not be of any use to applications.

6.1.13 DHS_ERR_LEVEL Error level of an error status

This is an enumerated value indicating the severity of a DHS library error. Note that these do not refer to errors detected by a server while executing an ICD 1c command, but only to errors detected by the DHS library. Possible values of this type are:

DHS_EL_SEVERE — Severe error, correct operation of the DHS library is not guaranteed after this type of error occurs.

DHS_EL_ERROR — The current operation has failed. The DHS library is still in a usable state. Errors with this level should not occur in normal operation.

DHS_EL_WARNING — A condition exists which may affect the future operation of the library, e.g. a resource is nearing critical levels.

DHS_EL_INFO — Informational status only. The current operation has succeeded. The library message string contains an informational message describing the operation.

DHS_EL_DEBUG — Message is a debugging message. Only the error callback routine should get messages of this type.

6.1.14 DHS_STATUS Status value returned by functions in the DHS library

This is an enumerated type indicating status in the DHS library. The status values are listed in Table 5 on page 18. Status codes that are prefixed by “DHS_E_” are error or severe status codes, other codes are prefixed by “DHS_S_”, and are warning or information codes. These status values may be returned in the status parameter of a function, or in the case of asynchronous errors or messages, they would be passed as an argument to the error callback function.

TABLE 5. DHS library status values

Status Symbol	Error level	Description
DHS_S_SUCCESS	**DHS_EL_INFO**	Normal completion of the function.
DHS_S_CMD_BULK_SENT	**DHS_EL_DEBUG**	Message sent as a IMP bulk message.
DHS_S_CMD_DELETE	**DHS_EL_DEBUG**	Deleting a command object.
DHS_S_CMD_MSG_SENT	**DHS_EL_DEBUG**	Sending a message.
DHS_S_CMD_NEW	**DHS_EL_DEBUG**	Creating a new command object.
DHS_S_CONNECT	**DHS_EL_INFO**	The error callback function of a server is called with this status when a client attempts to connect.
DHS_S_DISCONNECT	**DHS_EL_INFO**	The error callback function of a server is called with this status when a client disconnects.
DHS_S_FN_ENTRY	**DHS_EL_DEBUG**	Entering a function.
DHS_S_FN_EXIT	**DHS_EL_DEBUG**	Exiting a function.
DHS_S_IMP_EVENT	**DHS_EL_DEBUG**	An IMP message has been read.
DHS_S_MUTEX_LOCK	**DHS_EL_DEBUG**	Locking a mutex.
DHS_S_MUTEX_UNLOCK	**DHS_EL_DEBUG**	Unlocking a mutex.
DHS_S_NO_ATTRIB	**a**	No attribute was found matching the *index* parameter specified in a call to function **dhsAvIndex**, or the *name* parameter in a call to function **dhsAvFind**.
DHS_S_NO_FRAME	a	No frame was found matching the specification.

TABLE 5. DHS library status values (Continued)

Status Symbol	Error level	Description
DHS_S_NO_MESSAGE	[a]	No current error message found in response to a call to function **dhsMessage**.
DHS_S_NO_RESP	[a]	No response was found matching the *respNum* parameter specified in a call to function **dhsResponseGet**.
DHS_S_RECONNECT	**DHS_EL_INFO**	This client has automatically reconnected to a server.
DHS_S_SHUTDOWN	**DHS_EL_INFO**	The command server wants to shut down. The client receiving this message should close its connection to the command server.
DHS_S_SYS_EVENT	**DHS_EL_DEBUG**	An IMP system message was received.
DHS_S_USER_EVENT	**DHS_EL_DEBUG**	An IMP user message was received.
DHS_E_AVLIST_ARRAY	**DHS_EL_ERROR**	Attempted to add an array of AV lists as an attribute in an AV list.
DHS_E_BT_FIND	**DHS_EL_SEVERE**	Internal error.
DHS_E_BT_NOT_FOUND	**DHS_EL_SEVERE**	Internal error.
DHS_E_CB_NULL	**DHS_EL_ERROR**	A callback function required to process a message was NULL.
DHS_E_CB_TYPE	**DHS_EL_ERROR**	An attempt was made to use function **dhsCallbackSet** to set a callback function of an unknown type.
DHS_E_CMD_DELETED	**DHS_EL_ERROR**	An attempt was made to reference a command that is scheduled for deletion.
DHS_E_CMD_LOST	**DHS_EL_ERROR**	A server disconnected before finishing the processing of a command.
DHS_E_CMD_FIND	**DHS_EL_ERROR**	Internal error.
DHS_E_CMD_NOT_FOUND	**DHS_EL_ERROR**	Could not find a required command structure.
DHS_E_CON_FIND	**DHS_EL_ERROR**	Could not find required connection structure.
DHS_E_CON_INVALID	**DHS_EL_ERROR**	Attempt to send to an invalid connection id.
DHS_E_CON_LOCKOUT	**DHS_EL_ERROR**	An attempt was made to connect to this server before it is ready to accept connections or after it has started to execute the **dhsExit** function.
DHS_E_CON_LOST	**DHS_EL_ERROR**	A connection to a server or client was lost unexpectedly. The connection is no longer valid.
DHS_E_CON_NOT_FOUND	**DHS_EL_ERROR**	Could not find a required connection structure.
DHS_E_COND	**DHS_EL_SEVERE**	An error was returned from the POSIX thread library while manipulating a condition variable.
DHS_E_CONNECT	**DHS_EL_ERROR**	An error occurred while attempting to connect to a server.
DHS_E_CTL_CMD	**DHS_EL_ERROR**	An ICD3 control command failed.
DHS_E_DISCONNECT	**DHS_EL_ERROR**	An invalid disconnect message was received.
DHS_E_EL_RUNNING	**DHS_EL_ERROR**	An attempt was made to run the event loop when the event loop was already running.
DHS_E_ERS	**DHS_EL_ERROR**	An error occurred in the DRAMA ERS library.
DHS_E_ERS_MSG	**DHS_EL_ERROR**	This is used to report an error message from the DRAMA ERS library.

TABLE 5. DHS library status values (Continued)

Status Symbol	Error level	Description
DHS_E_IMP	**DHS_EL_ERROR**	An error occurred in the IMP library.
DHS_E_IMP_MSG	**DHS_EL_ERROR**	An invalid IMP message was received.
DHS_E_IMP_REGISTER	**DHS_EL_SEVERE**	An error occurred while attempting to register this task as an IMP task.
DHS_E_IMP_SYS_MSG	**DHS_EL_ERROR**	An unknown IMP system message was received.
DHS_E_IMP_USER_MSG	**DHS_EL_ERROR**	An unknown IMP user message was received.
DHS_E_INIT	**DHS_EL_SEVERE**	An attempt was made to use the DHS library before calling function **dhsInit**.
DHS_E_LOCATE	**DHS_EL_ERROR**	IMP failed to located the requested task during a connection.
DHS_E_LOCATE_MSG	**DHS_EL_ERROR**	An IMP locate system message was received for a connection that was not waiting to be connected.
DHS_E_LOOP_TYPE	**DHS_EL_ERROR**	Attempt to run the event loop with an invalid event loop type.
DHS_E_MEMORY	**DHS_EL_SEVERE**	A memory allocation failure has occurred.
DHS_E_MSG_LENGTH	**DHS_EL_ERROR**	A message was received with the wrong length.
DHS_E_MUTEX	**DHS_EL_SEVERE**	An error was returned by the POSIX thread library while manipulating a mutex.
DHS_E_NO_ATTRIB	**DHS_EL_ERROR**	A required attribute was not found in a command.
DHS_E_NOT_AVLIST	**DHS_EL_ERROR**	The specified SDS structure was not an AV list.
DHS_E_NULLVALUE	**DHS_EL_ERROR**	An attribute had a null value where a null value is not permitted.
DHS_E_PTR_SIZE	**DHS_EL_ERROR**	The size of a pointer does not match the expected size.
DHS_E_SDS	**DHS_EL_SEVERE**	An error occurred in an SDS routine.
DHS_E_TASK_UNKNOWN	**DHS_EL_ERROR**	An attempt was made to locate a task that was not known to IMP.
DHS_E_THREAD_CREATE	**DHS_EL_ERROR**	Failed to create a thread.
DHS_E_TSD	**DHS_EL_SEVERE**	An error occurred while attempting to access the thread specific data structure.
DHS_E_TYPE	**DHS_EL_TYPE**	An unknown data type was specified.

a. These status values are not associated with a message, and so do not have an associated error level.

6.1.15 DHS_TAG — Identifies a specific command

This value is used to identify a specific command in progress. It is returned from the **dhsApply** function, is used as input to functions that must identify a particular command, and is passed to the callback functions to allow an application to identify a command.

6.1.16 DHS_THREAD — Identifies a thread.

Under Solaris this value will be of type pthread_t (defined by the POSIX thread library) and will be the thread id returned by the pthread_create function. Under VxWorks this will be an int, and will be the value returned by the VxWorks **taskSpawn** function.

6.2 General functions

6.2.1 dhsInit — Initialize the DHS library

```
void                  dhsInit
(
    const char        *name,         /* in    : Name to register with IMP.           */
    int               numConnect,    /* in    : Maximum number of connections.       */
    DHS_STATUS        *status        /* mod   : Function return status.              */
)
```

This function initializes the ICD 1c library. This function must be called before any other function in the DHS library. It is an error to call this function more than once, unless the **dhsExit** function is called between invocations. The *name* parameter specifies the name to be used by IMP to locate this UNIX process or VxWorks task. Each UNIX process or VxWorks task can only be known by one name. The *numConnect* argument specifies the maximum number of connections to be made or accepted by the task.

6.2.2 dhsExit — Clean up the DHS library

```
void                  dhsExit
(
    DHS_STATUS        *status        /* mod   : Function return status.              */
)
```

This function closes any open connections, stops the event loop, and frees up any resources allocated to the library. No other DHS library functions should be called after this function is called unless the **dhsInit** function is called again.

6.2.3 dhsMessage — Get the DHS library error numbers and level, and a pointer to the message string

```
char *                dhsMessage
(
    DHS_STATUS        *errorNum,     /* out   : DHS library error number.            */
    DHS_ERR_LEVEL     *errorLev,     /* out   : DHS library error level.             */
    DHS_STATUS        *status        /* mod   : Function return status.              */
)
```

The DHS library maintains thread specific queues of library messages describing errors in the library functions. The **dhsMessage** function returns a pointer to the oldest message in the queue of messages assigned to the current thread. If more than one message is in the queue, subsequent messages may be accessed by calling **dhsMessageClear** and **dhsMessage** until the *status* parameter is set to **DHS_S_NO_MESSAGE**. Note that an exception to the thread specific messages occurs when errors related to accessing thread specific data occur. The *msg* pointer may be set to NULL. Error messages should only be deleted with the **dhsMessageClear** function, and the pointer returned by this function should **NOT** be freed.

Note that this function only returns the messages reporting failure of DHS library functions, not messages describing the success of failure of commands issued by the DHS library. Command status can be retrieved with the **dhsStatus** function.

6.2.4 dhsMessageClear — Clear a DHS message

```
void                  dhsMessageClear
(
    DHS_STATUS        *status        /* mod   : Function return status.              */
)
```

This function removes the oldest message from the message queue for the current thread. See also function **dhsMessage**. If **dhsMessageClear** is called with no messages in the queue, the *status* parameter will be set to **DHS_S_NO_MESSAGE**.

6.2.5 dhsEventLoop — Start the DHS event loop

```
void                    dhsEventLoop
(
    DHS_EL_TYPE         loopType,       /* in    : Type of event loop to run.            */
    ...,                                /* in    : Args depend on loopType               */
    DHS_STATUS          *status         /* out   : Function return status.               */
)
```

This function starts the DHS event loop. The event loop may be run in various ways, selected by the value of *loopType*. The argument list of **dhsEventLoop** depends on the type of event loop being run. The following function descriptions detail how **dhsEventLoop** would be used with the various values of *loopType*. In all variations of the function, the event loop can be stopped by calling function **dhsEventLoopEnd**. All variations of the function will return an error if the event loop is already running.

6.2.5.1 dhsEventLoop — Start the DHS event loop in a thread

```
void                    dhsEventLoop
(
    DHS_EL_TYPE         loopType = DHS_ELT_THREADED,
                                        /* in    : Type of event loop to run.            */
    DHS_THREAD          *threadId       /* out   : Id of the thread created.             */
    DHS_STATUS          *status         /* mod   : Function return status.               */
)
```

This variation of function **dhsEventLoop** causes the DHS library to create a new thread for the event loop and then return to the calling program. When the event loop is terminated by calling function **dhsEventLoopEnd** the created thread will terminate. The id of the thread created is returned in the threadId parameter which can be passed NULL if you don't care what the thread id is.

6.2.5.2 dhsEventLoop — Start the DHS event loop and wait indefinitely

```
void                    dhsEventLoop
(
    DHS_EL_TYPE         loopType = DHS_ELT_WAIT,
                                        /* in    : Type of event loop to run.            */
    DHS_STATUS          *status         /* mod   : Function return status.               */
)
```

This variation of **dhsEventLoop** runs the event loop in the current thread. The function will not return until the event loop is terminated by calling **dhsEventLoopEnd**, at which time the function will return to the calling program. There are no additional arguments in this variation of the function.

6.2.5.3 dhsEventLoop — Process all pending events and then return

```
void                    dhsEventLoop
(
    DHS_EL_TYPE         loopType = DHS_ELT_NO_WAIT,
                                        /* in    : Type of event loop to run.            */
    DHS_STATUS          *status         /* mod   : Function return status.               */
)
```

This variation of **dhsEventLoop** processes all pending events and then returns to the calling program. There are no additional arguments in this version of the function. This function is required internally to the DHS library, and it may not be useful to applications.

6.2.5.4 dhsEventLoop — Process events until some condition is met

```
void                    dhsEventLoop
(
    DHS_EL_TYPE         loopType = DHS_ELT_COND,
                                       /* in    : Type of event loop to run.            */
    DHS_COND_FN         condFn,        /* in    : Condition test function.              */
    void                *condArg,      /* in    : Test function argument.               */
    DHS_STATUS          *status        /* mod   : Function return status.               */
)
```

This variation of **dhsEventLoop** runs the event loop using the current thread until a condition is met. The condition is tested by calling the function specified in argument *condFn*. The additional arguments in this variation of the function are:

- *condFn*: Pointer to a function that takes a void pointer as an argument and returns a boolean value.
- *condArg*: Void pointer to be passed as the argument to function *condFn*.

This function is required internally to the DHS library, and may not be useful to applications. Calling function **dhsEventLoopEnd** will cause this function to return before the condition is met.

6.2.6 dhsEventLoopEnd — Stop the event loop

```
void                    dhsEventLoopEnd
(
    DHS_STATUS          *status        /* mod   : Function return status.               */
)
```

This function causes an event to be added to the IMP event list, which when interpreted by the event loop causes the event loop to stop running. If the event loop is running in a thread created by the DHS library, the thread will terminate. If the event loop is running in an application thread, then the **dhsEventLoop** function will return.

6.2.7 dhsCallbackSet — Set a DHS library callback function

```
void                    dhsCallbackSet
(
    DHS_CB_TYPE         cbType,        /* in    : Type of callback function.            */
    DHS_CB_FN_PTR       cbFn,          /* in    : The callback function.                */
    DHS_STATUS          *status        /* mod   : Function return status.               */
)
```

This function sets or replaces a callback function for the DHS library. If the callback function pointer is NULL, the existing callback is removed. The callback functions are described in other function descriptions. The parameters of the callback functions vary depending on their purpose. The type of callback function being set is determined by the value of the *cbType* parameter. The types of callback function that can be set are:

- **DHS_CBT_ERROR**: Set the callback function to be called when the DHS library event loop detects an error or wants to display an information or debug message.
- **DHS_CBT_RESPONSE**: Set the callback function to be called by the DHS library when a command response is received on a client system.
- **DHS_CBT_COMMAND**: Set the callback function to be called by the DHS library when a command is received on a server system.
- **DHS_CBT_CONNECT**: Set the callback function to be called by the DHS library when the state of a connection changes.

Callback functions can be set at any time, however they will normally be set during the initialization of the application. Setting a callback function to NULL disables the callback function for the callback function type specified.

6.2.8 dhsDisconnect Disconnect from a server

```
void                    dhsDisconnect
(
    DHS_CONNECT         connection,    /* in    : The connection to close.              */
    DHS_STATUS          *status        /* mod   : Function return status.               */
)
```

This function is used by a client to break the connection to an ICD 1c server, or used by a server to refuse a connection request from a client. The connection in the *connection* parameter is not valid after this function is called.

6.2.9 dhsDebugLevel Set the debugging level of the DHS library

```
void                    dhsDebugLevel
(
    DHS_DEBUG_LEVEL     level,         /* in    : The new debugging level.              */
    DHS_STATUS          *status        /* mod   : Function return status.               */
)
```

This function sets the debugging level for the DHS library. Debug messages are reported via the DHS library error callback function. Valid debug levels are described in Section 6.1.11 on page 17.

6.2.10 dhsElBroadcast Cause the event loop to execute on no-op iteration

```
void                    dhsElBroadcast
(
    DHS_STATUS          *status        /* mod   : Function return status.               */
)
```

This function can be used in the rare cases where it is necessary to cause the DHS event loop to do an iteration. This could be used if a client wanted to change the state of a command and then cause the DHS event loop to stop waiting for a response to the command.

6.2.11 dhsIsConnected Check a connection to see if it is still connected

```
DHS_BOOLEAN             dhsIsConnected
(
    DHS_CONNECT         connection,    /* in    : The connection to check.              */
    DHS_STATUS          *status        /* mod   : Function return status.               */
)
```

This function is called to test the state of a connection. The function returns **DHS_TRUE** if the connection is open and ready to use, **DHS_FALSE** otherwise.

6.3 Client functions

The following functions are expected to be used primarily by client systems.

6.3.1 **dhsConnect** Connect to a server

```
DHS_CONNECT            dhsConnect
(
    const char         *ipAddress,    /* in    : Servers internet address.             */
    const char         *name,         /* in    : Name of the server.                   */
    void               *userData,     /* in    : User data pointer for the connection. */
    DHS_STATUS         *status        /* mod   : Function return status.               */
)
```

This function initiates a connection to an ICD 1c server. If successful, the connection id is returned as the value of the function. The *name* parameter indicates the name of the server, which must be the same as the name specified in the *name* parameter the server used when it called the **dhsInit** function. The userData parameter points to a user defined structure associated with the connection, and can be retrieved with the **dhsConUserDataGet** function. If not required this value can be NULL. The user data pointer is described in Section 5.1 on page 11.

6.3.2 **dhsConUserDataGet** Get a connection's user data pointer

```
void                   *dhsConUserDataGet
(
    DHS_CONNECT        connection,    /* in    : The connection.                       */
    DHS_STATUS         *status        /* mod   : Function return status.               */
)
```

This function returns the user data pointer associated with a connection.

6.3.3 **dhsConUserDataSet** Set a connection's user data pointer

```
void                   dhsConUserDataSet
(
    DHS_CONNECT        connection,    /* in    : The connection.                       */
    void               *userData,     /* in    : The new user data pointer.            */
    DHS_STATUS         *status        /* mod   : Function return status.               */
)
```

Set the user data pointer associated with a connection. The user data pointer argument in the **dhsConnect** function is the preferred way of setting the user data pointer. Since connection state changes are asynchronous events, when this function is used it is possible that a callback function could be executing before or while the user data pointer is being updated, and so it is necessary to protect the changes and use of the user data pointer with thread synchronization mechanisms (i.e. mutexes).

6.3.4 **dhsConnectInfo** Returns information about the other end of a connection

```
void                   dhsConnectInfo
(
    DHS_CONNECT        connection,    /* in    : The connection to query.              */
    char               **ipAddress,   /* out   : Returns a pointer to the IP address.  */
    char               **name,        /* out   : Returns the IMP name.                 */
    DHS_STATUS         *status        /* mod   : Function return status.               */
)
```

This function returns information about the system at the other end of a connection.

6.3.5 **dhsApply** Sends a command to a server

```
DHS_TAG                dhsApply
(
    DHS_CONNECT        connection,    /* in    : Server connection.                    */
    const char         *command,      /* in    : The command name.                     */
    DHS_AV_LIST        avList,        /* in    : AV list for the command.              */
    void               *userData,     /* in    : User data pointer.                    */
    DHS_STATUS         *status        /* mod   : Function return status.               */
)
```

This function is used to send a command to an ICD 1c server. The connection to the server specified in parameter *connection* must have been established with a call to dhsConnect. The attribute value list specified by parameter *avList* must have been created with functions **dhsAvListNew** and **dhsAvAdd** (DHS_AV_LIST_NULL is valid as the value of *avList* if there are no attributes associated with a command). The *userData* parameter can be used by an application to associate a pointer with the command tag, as described in Section 5.1 on page 11 (NULL can be specified for the user data pointer if it isn't being used). The **DHS_TAG** returned by this function allows the command to be identified at a later point in the processing of the command, for example to abort the command, or to identify the command associated with a response. It is the responsibility of the application program to use function **dhsTagFree** to free the tag **after** the command action has reached one of the end states defined in Figure 3 on page 8.

Note that an application using a response callback function to handle responses to a command should prepare itself to receive responses **before dhsApply** is executed. Because of thread synchronization considerations, there is no guarantee the thread executing the dhsApply function will even have time to return from the dhsApply function before responses start arriving. This problem can also be dealt with using thread synchronization mechanisms (i.e. mutexes).

The status return value from this function only indicates the success of this function, not the status of the resulting command. If the **dhsApply** function succeeds, then the status of the command can be queried by passing the returned **DHS_TAG** to the **dhsStatus** function. If the **dhsApply** function fails then no command is created, the returned value will be **DHS_TAG_NULL**, and there is nothing for **dhsStatus** to query.

6.3.6 dhsWait — Wait for a list of command actions to complete

```
void                dhsWait
(
    int             numTags,       /* in    : Number of tags in tagList.          */
    const DHS_TAG   tagList*,      /* in    : A list of command tags.             */
    DHS_STATUS      *status        /* mod   : Function return status.             */
)
```

This command waits until all commands listed in the *tagList* parameter have reached a terminal state (i.e. **DHS_CS_ABORTED**, **DHS_CS_DONE**, **DHS_CS_ERROR,** or **DHS_CS_LOST**). This function will exit before the command actions are complete if function **dhsEventLoopEnd** is called. If a response callback function has been set, the response callback function is guaranteed to be finished executing before this function returns.

6.3.7 dhsStatus — Query the current status of a command

```
DHS_CMD_STATUS      dhsStatus
(
    DHS_TAG         tag,           /* in    : tag of the command to query.        */
    char            **string,      /* out   : Description of the status.          */
    DHS_STATUS      *status        /* mod   : Function return status.             */
)
```

This function returns the current status of a command. This function does not query the server, but simply returns the last status reported by the server and stored by the DHS library. Note that the status returned by this function is the status of a command or bulk data request, not the status of the function that initiated the command or bulk data request (**dhsApply** from ICD 1c, or **dhsBdGet**, and **dhsBdGet** from ICD 3). If the initiating function returns an error status, then no command or bulk data request is created, and therefore there is nothing for this function to query.

Note that if this function indicates that the command failed, a textual description of the failure is located in the string parameter. The **dhsMessage** function will **not** return any information about the failure of the command, since the **dhsMessage** function only returns information about the failure of **DHS library** functions, not of the commands it initiates.

6.3.8 dhsTagFree — Free a command tag

```
void                  dhsTagFree
(
    DHS_TAG           tag,           /* in   : The tag to free.                  */
    DHS_STATUS        *status        /* mod  : Function return status.           */
)
```

This function must be used by an application which has called **dhsApply** in order to free the **DHS_TAG** structure. This must only be done **after** the command has reached a terminal state.

6.3.9 dhsUserDataGet — Get the user data pointer for a command

```
void                  *dhsUserDataGet
(
    DHS_TAG           tag,           /* in   : Command tag.                      */
    DHS_STATUS        *status        /* mod  : Function return status.           */
)
```

Return the user data pointer associated with a command. The user data pointer is described in Section 5.1 on page 11.

6.3.10 dhsUserDataSet — Set the user data pointer for a command

```
void                  dhsUserDataSet
(
    DHS_TAG           tag,           /* in   : Command tag.                      */
    void              *userData,     /* in   : The new user data pointer.        */
    DHS_STATUS        *status        /* mod  : Function return status.           */
)
```

This function allows the user data pointer associated with a command to be set. The user data parameter of the **dhsApply** function is the preferred way of setting the user data pointer. This function should be used with extreme caution, and the setting and use of the user data pointer should be protected by synchronization mechanisms (i.e. mutexes) to avoid the problems caused if the command response callback function is executed before or during the update of the user data pointer.

6.3.11 dhsResponseGet — Get the response associated with a command

```
DHS_AV_LIST           dhsResponseGet
(
    DHS_TAG           tag,           /* in   : Command tag.                      */
    int               respNum,       /* in   : The response number to get.       */
    DHS_STATUS        *status        /* mod  : Function return status.           */
)
```

This function is supplied to allow applications that don't use callback functions to access the command responses. If the command results in multiple responses, the parameter *respNum* can be used to indicate which response should be retrieve (0 is the first response). If there is no response corresponding to the respNum parameter, this function will set the *status* parameter to **DHS_S_NO_RESP**.

6.3.12 dhsTagDone — Check to see if a command tag is in one of the end states

```
DHS_BOOLEAN           dhsTagDone
(
    DHS_TAG           tag,           /* in   : Command tag.                      */
    DHS_STATUS        *status        /* mod  : Function return status.           */
)
```

This function is called to check to see if a command is complete. The function returns DHS_TRUE if the command is complete, or DHS_FALSE if it is not complete.

6.4 Server functions

These functions are expected to be used primarily by ICD 1c servers, e.g. the DHS command server and DHS data server.

6.4.1 dhsCmdResponse Send a response to a command

```
void                  dhsCmdResponse
(
    DHS_CONNECT       connection,    /* in   : Connection to the client.          */
    DHS_TAG           tag,           /* in   : Client command tag.                */
    DHS_CMD_STATUS    status,        /* in   : New status for the command.        */
    char              *statusString, /* in   : New status description.            */
    DHS_AV_LIST       avList,        /* in   : Response AV list.                  */
    DHS_STATUS        *status        /* mod  : Function return status.            */
)
```

This function is used by a server to send a response to a client. The *avList* parameter may be set to NULL if only the command status and status strings are to change. The *statusString* parameter may be set to NULL if no string is to be associated with the command status. The valid command states are listed in Section 6.1.6 on page 16.

The sequence of command states set with **dhsCmdResponse** must follow the permitted sequences shown in Figure 3 on page 8.

6.4.2 dhsServerExit Request all clients close their connections

```
void                  dhsServerExit
(
    DHS_STATUS        *status        /* mod  : Function return status.            */
)
```

This function sends a message to all client systems requesting that they close their connections. The error callback routine on each of the client systems will be executed with a status of **DHS_S_SHUTDOWN**. This function will return after all connected clients have finished processing the error callback.

6.4.3 dhsSetTimeout Set the amount of time a server expects to take to restart

```
void                  dhsSetTimeout
(
    int               timeout        /* in   : The timeout interval in seconds.   */
)
```

This function sets the number of seconds a client should wait for this server to restart if a connection should be lost. This value is sent to all clients when they connect to the server, and the DHS library in the client system will automatically attempt to re-establish a connection to this server for the specified interval. The purpose of this timeout is to allow a server to indicate how long it thinks it will take to reset, and therefore allows clients to automatically re-establish connections after a reset. If this function isn't called, the default timeout is 10 seconds.

6.5 Attribute value list manipulation functions

These functions will be used by both clients and servers to create and interpret attribute value lists.

6.5.1 dhsAvListNew Allocate a new attribute value list

```
DHS_AV_LIST           dhsAvListNew
(
    DHS_STATUS        *status        /* mod  : Function return status.            */
)
```

This function allocates a new empty attribute value list.

6.5.2 dhsAvListFree Free an attribute value list

```
void                  dhsAvListFree
(
    DHS_AV_LIST       avList,       /* in   : The AV list to free.                  */
    DHS_STATUS        *status       /* mod  : Function return status.               */
)
```

This function frees an attribute value list. This function should only be used to free attribute value lists created by an application with the **dhsAvListNew** function, or by a server to free the AV list passed as an argument to the command callback function.

6.5.3 dhsAvAdd Add an attribute and its value to an AV list

```
void                  dhsAvAdd
(
    DHS_AV_LIST       avList,       /* in   : The AV list to be extended.           */
    const char        *attName,     /* in   : The new attribute name.               */
    DHS_DATA_TYPE     attType,      /* in   : The type of the attribute.            */
    int               ndims,        /* in   : Number dimensions of the attribute.   */
    const unsigned long *dims,      /* in   : Vector containing the size of each    */
                                    /*      dimension.                              */
    ?                 value,        /* in   : The value of the attribute.           */
    DHS_STATUS        *status       /* mod  : Function return status.               */
)
```

This function allows an new attribute to be added to an attribute value list. The function is implemented as a vararg function, where the number and type of arguments depends on the *attType* and the *ndims* parameters. The type of the *value* parameter depends on the *attribType* and *ndims* parameters. If the *ndims* parameter is 0, the value is assumed to be a scalar value, the *dims* parameter can be NULL (it is ignored) and the *value* parameter should be the actual data item. If the *ndims* parameter is greater than zero, the size of each dimension should be indicated in the *dims* parameter, and the *value* parameter should be a pointer to the data item. See Section 7.2 on page 34 for examples of the use of this function.

6.5.4 dhsAvDelete Delete an attribute and its value from an AV list

```
void                  dhsAvDelete
(
    DHS_AV_LIST       avList,       /* in   : The AV list to be changed.            */
    const char        *attName,     /* in   : The name of the attribute to delete.  */
    DHS_STATUS        *status       /* mod  : Function return status.               */
)
```

This function deletes an attribute with the name *attName* from the attribute value list with id *avList*. If the attribute cannot be found, a status of DHS_S_NO_ATTRIB is returned. Note that this function will change the index numbers used by function **dhsAvIndex** for all attributes in *avList* where the index is greater than the index of the deleted attribute.

6.5.5 dhsAvInfo Get information about an attribute

```
void                  dhsAvInfo
(
    DHS_AV_ID         attrib,       /* in   : Attribute id to query.                */
    char              **name,       /* out  : The name of the attribute.            */
    DHS_DATA_TYPE     *type,        /* out  : The type of the attribute.            */
    int               *ndims,       /* out  : Number of dimensions of the attribute. */
    unsigned long     *dims,        /* out  : Array to contain size of each dimension.*/
    void              **value,      /* out  : Pointer to the value.                 */
    DHS_STATUS        *status       /* mod  : Function return status.               */
)
```

This command gets information about an attribute from an attribute value list. The *attrib* parameter indicates which attribute to query. If there is no item corresponding to *attrib*, the function will return the status **DHS_S_NO_ATTRIB**. The *dims* parameter indicates the dimensions of the attribute value. The *dims* parameter must be an array of 7 integers to allow it to contain the maximum number of dimensions supported by SDS.

6.5.6 dhsAvFind Find an attribute by name

```
DHS_AV_ID               dhsAvFind
(
    DHS_AV_LIST         avList,        /* in    : The AV list to query.                 */
    const char          *name,         /* in    : Attribute to find.                    */
    DHS_STATUS          *status        /* mod   : Function return status.               */
)
```

This function locates an attribute in an attribute value list by name and returns its id. Function **dhsAvInfo** can then be used to get any other information about the attribute. If no attribute is found matching the *name* parameter, the function will return with status **DHS_S_NO_ATRRIB**.

6.5.7 dhsAvIndex Find an attribute based on its position in the AV list

```
DHS_AV_ID               dhsAvIndex
(
    DHS_AV_LIST         avList,        /* in    : The AV list to query.                 */
    int                 index,         /* in    : The position in the avList.           */
    DHS_STATUS          *status        /* mod   : The function return status.           */
)
```

This function finds an attribute based on position in an AV list. The *index* parameter indicates where the attribute is in the list, where 0 indicates the first item in the list. If no attribute can be found matching the index number, the function will return with status **DHS_S_NO_ATTRIB**.

6.5.8 dhsAvListAccess Access an AV list in a memory buffer

```
DHS_AV_LIST             dhsAvListAccess
(
    const void          *buffer,       /* in    : The buffer to access.                 */
    DHS_STATUS          *status        /* mod   : Function return status.               */
)
```

This function makes a memory buffer created by the **dhsAvListExport** function available as a read-only AV list. This is primarily a debugging/simulation support function. The AV list returned by this function should be freed with the **dhsAvListFree** function when it is no longer needed. Note that this function creates an AV list that points to data in the buffer, so it is necessary to keep the buffer intact for as long as the AV list is needed. If a stand alone, writable AV list is required, a copy can be made with the **dhsAvListCopy** function.

6.5.9 dhsAvListCopy Copy an attribute value list

```
DHS_AV_LIST             dhsAvListCopy
(
    DHS_AV_LIST         avList,        /* in    : The AV list to copy.                  */
    DHS_STATUS          *status        /* mod   : Function return status.               */
)
```

This function creates a new attribute value list containing a copy of the contents of an existing AV list. This function should be used sparingly, especially on large AV Lists, since it isn't very efficient. The new attribute value list must be freed with the **dhsAvListFree** function when it is no longer needed.

6.5.10 dhsAvListExport — Export an attribute value list to a memory buffer

```
void                 dhsAvListExport
(
    DHS_AV_LIST          avList,        /* in    : The AV list to export.              */
    void                 *buffer,       /* mod   : The buffer to receive the av list.  */
    unsigned int         bufSize,       /* in    : The size of buffer.                 */
    DHS_STATUS           *status        /* mod   : The function return status.
)
```

This function exports an AV list to a memory buffer. The **dhsAvListAccess** function can then be used to make the buffer usable as an AV list again (usually after writing it to disk, or sending it as a message, or some other thing that can only be done to a contiguous memory buffer). The buffer must be allocated before calling this function, and must be large enough to contain the exported structure. The expected size of the exported structure can be determined with the **dhsAvListSize** function. This is primarily a debugging/simulation support function.

6.5.11 dhsAvListPrint — Print an AV list to standard out

```
void                 dhsAvListPrint
(
    DHS_AV_LIST          avList,        /* in    : The AV list to print.               */
    DHS_STATUS           *status        /* mod   : The function return status.         */
)
```

This function prints the contents of the AV list to standard output in a readable form. Because of its limitations, this function is only useful for debugging.

6.5.12 dhsAvListSize — Return the size of an AV list

```
unsigned long        dhsAvListSize
(
    DHS_AV_LIST          avlist,        /* in    : The AV list to size.                */
    DHS_STATUS           *status        /* mod   : Function return status.             */
)
```

This function determines the amount of space an AV list would occupy if it were exported to a memory buffer. This function is used prior to calling **dhsAvListExport** to determine how big the buffer must be.

6.6 User supplied functions

These functions are supplied by applications using the ICD 1c interface, and are set using the **dhsCallbackSet** function. The names of these functions are not significant.

6.6.1 error callback — User supplied error callback function

```
void                 error_callback
(
    DHS_CONNECT          connect,       /* in    : Connection causing the error.       */
    DHS_STATUS           errorNum,      /* in    : The error number.                   */
    DHS_ERR_LEVEL        errorLev,      /* in    : The error level.                    */
    char                 *msg,          /* in    : The error message string.           */
    DHS_TAG              tag,           /* in    : The command tag of the error.       */
    void                 *userData      /* in    : The user data associated with an error.*/
)
```

This is an application supplied callback function to handle event loop errors. The function is set by calling function **dhsCallbackSet** with **DHS_CBT_ERROR** as the value of the *cbType* parameter. The library will automatically call function **dhsMessageClear** to remove the error from the error queue. See Table 5 on page 18 for a list of error symbols and their levels. If the error is associated with a command, the *tag* and *userData* parameters will be set to the DHS_TAG and user data pointers for the command. If no command is associated with the error, the *tag* parameter will be set to **DHS_TAG_NULL** and the user data pointer will be NULL.

6.6.2 **response callback** User supplied command response callback function

```
void                  response_callback
(
    DHS_CONNECT       connection,    /* in   : Connection for the command.          */
    DHS_TAG           tag,           /* in   : The command tag.                     */
    char              *command,      /* in   : The command string.                  */
    DHS_CMD_STATUS    status         /* in   : Current status of command.           */
    char              *string,       /* in   : Description of the status.           */
    DHS_AV_LIST       avList,        /* in   : Command response AV list.            */
    void              *userData      /* in   : The user data pointer.               */
)
```

This is a application supplied callback function to handle command responses for a client application. The function is set by calling function **dhsCallbackSet** with **DHS_CBT_RESPONSE** as the value of the *cbType* parameter. The *avList* parameter may be **DHS_AV_LIST_NULL** if the server has updated the status without setting a response AV list. The *string* parameter may be NULL if no character string was assigned to the status by the command server. The AV list in parameter *avList* should **not** be freed in the response callback. This AV list is a part of the tag, and will be freed when the tag is freed. When a command reaches a final state (i.e. command status is **DHS_CS_ABORTED**, **DHS_CS_DONE**, **DHS_CS_ERROR**, or **DHS_CS_LOST**), the application is responsible for freeing the tag associated with the command. This may be done from within this callback function.

6.6.3 **command callback** User supplied command callback function

```
void                  command_callback
(
    DHS_CONNECT       connection,    /* in   : Connection to the client.            */
    DHS_TAG           tag,           /* in   : The client command tag.              */
    char              *command,      /* in   : The name of the command.             */
    DHS_AV_LIST       avList         /* in   : The command AV list.                 */
)
```

This is a server supplied callback function to handle commands for a server application. The name of this function is not significant, the server can use any name. The function is set by calling function **dhsCallbackSet** with **DHS_CBT_COMMAND** as the value of the *cbType* parameter. The *avList* parameter may be **DHS_AV_LIST_NULL** if the client has not supplied an AV list for the command. The AV list in parameter *avList* will be feed automatically when the command callback function returns. The DHS library will not process new events until this function returns, so it may be necessary to create a thread to handle slow commands, in which case it will be necessary to make a copy of *avList* using function **dhsAvListCopy** (if the AV list is to be used in the new thread).

6.6.4 **connection callback** User supplied connection callback function

```
void                  connect_callback
(
    DHS_CONNECT       connection,    /* in   : The connection that changed state.   */
    DHS_CON_STATE     state          /* in   : The new state of the connection.     */
)
```

This is a user supplied callback function which is executed when a connection changes state. The name of this function is not significant. The function is set by calling function **dhsCallbackSet** with **DHS_CBT_CONNECT** as the value of the *cbType* parameter. This function does not have to be set. The function can be set for either client or server applications.

6.7 Data types

The data types listed in Table 6 on page 33 are used to construct attribute value lists. The type names are members of the **DHS_ATT_TYPE** enumerated data type.

TABLE 6. Primitive data types

Type Name	Description	Range
DHS_DT_CHAR	Any ASCII character.	N/A
DHS_DT_INT8	Eight bit integer value.	-127 to 127
DHS_DT_UINT8	Unsigned eight bit integer value.	0 to 255
DHS_DT_INT16	Sixteen bit integer value.	-32767 to 32767
DHS_DT_UINT16	Unsigned sixteen bit integer value.	0 to 65535
DHS_DT_INT32	Thirty two bit integer value.	-2147483647 to 2147483647
DHS_DT_UINT32	Unsigned thirty two bit integer value.	0 to 4294967295
DHS_DT_FLOAT	Thirty two bit floating point value.	machine dependent
DHS_DT_DOUBLE	Sixty four bit floating point value.	machine dependent
DHS_DT_STRING	Null terminated ASCII string.	N/A
DHS_DT_TAG	Command tag identifier.	N/A
DHS_DT_AVLIST	An attribute value list.	N/A
DHS_DT_BOOEAN	A logical value that can be **DHS_TRUE or DHS_FALSE.**	N/A
DHS_DT_NONE	No data type. Used when there is no data present in an item.	N/A

7.0 Example Code

The following sections illustrate typical uses of the library. The error checking has been minimized in the following examples to simplify them.

7.1 Initialization of a client application

Following is an example of the code that would normally be used to initialize a client application. It is assumed that the client will use callbacks to handle library events. The user supplied functions in this code fragment are:

my_error_callback — Callback function executed when asynchronous DHS library errors occur.

my_response_callback — Callback function executed when command responses are received.

my_connect_callback — Callback function executed when a connection changes state.

panic — Executed when a DHS function fails.

```
#include “dhs.h”
main( int argc, char ** argv)
{
       void          my_error_callback(); /* Application supplied error callback.    */
       void          my_response_callback();
                                          /* Application supplied response callback.*/
       void          my_connect_callback();
       DHS_CONNECT   connect;             /* Connection to the command server.       */
       DHS_STATUS    status;              /* DHS library status value.               */
```

```
        status = DHS_S_SUCCESS;


        /*
         * Initialize the DHS library with space for 10 connections.
         */

        dhsInit( “my_unique_name”, 10, &status );


        /*
         * Set up the callback functions.
         */

        dhsCallbackSet( DHS_CBT_ERROR, my_error_callback, &status );
        dhsCallbackSet( DHS_CBT_RESPONSE, my_response_callback, &status );
        dhsCallbackSet( DHS_CBT_CONNECT, my_connect_callback, &status );


        /*
         * Start the event loop in its own thread.
         */

        dhsEventLoop( DHS_ELT_THREADED, NULL, &status );


        /*
         * Make connections to the DHS command server.
         */

        connect = dhsConnect( “DHScontrol.north.gemini.edu”, “dhsCommand”, NULL, &status );


        /*
         * Check the status to ensure everything worked.
         */

        if ( status != DHS_S_SUCCESS )
        {
               panic();
        }


        /*
         * Do other client application stuff.
         */
}
```

7.2 Sending a command to a server

Following is an example of the code required to generate an attribute value list and execute a command. This program assumes the existence of the following functions:

panic — This function is executed when an error occurs.

```
#include “dhs.h”

void    execCmd
(
        DHS_CONNECT    dhsServer,     /* The server to send the command to.            */
        void           *user_data_pointer
                                      /* Pointer to the user data for the command.     */
)
{
        short          ivalue;        /* A sample integer.                             */
        short          iarray[10][20];/* A sample array of integers.                   */
        char           *str_array[5];/* A sample array of strings.                    */
        DHS_AV_LIST    avList;        /* The attribute value list.                     */
        DHS_TAG        dhsTag;        /* The tag referring to the command.             */
        DHS_STATUS     status;        /* DHS library status value.                     */
        unsigned long  dims[2];       /* The dimensions of the data array.             */


        status = DHS_S_SUCCESS;
```

```
        /*
         * Create a new attribute value list.
         */

        avList = dhsAvListNew( &status );


        /*
         * Add an integer scalar value to the attribute value list.
         */

        dhsAvAdd( avList, “AttribName1”, DHS_DT_INT16, 0, NULL, ivalue, &status );


        /*
         * Add a 10 x 20 integer array to the attribute value list.
         */

        dims[0] = 10;
        dims[1] = 20;
        dhsAvAdd( avList, “AttribName2”, DHS_DT_INT16, 2, dims, iarray, &status );


        /*
         * Add an array of 5 null terminated strings to the attribute value list.
         */

        dims[1] = 5;
        dhsAvAdd( avList, “AttribName3”, DHS_DT_STRING, 1, dims, str_array, &status );


        /*
         * Add a string to the attribute value list.
         */

        dhsAvAdd( avList, “AttribName4”, DHS_DT_STRING, 0, NULL, “example string”,
                      &status );


        /*
         * Send the command to the server.
         */

        dhsTag = dhsApply( dhsServer, “example_command”, avList, user_data_pointer,
                      &status );


        /*
         * Free the attribute value list.
         */

        dhsAvListFree( avList, &status );


        /*
         * Check the status to make sure everything worked.
         */

        if ( status != DHS_S_SUCCESS )
        {
                panic();
        }
}
```

7.3 Client command response callback function

Following is an example of a client response callback function. It is assumed that the user data pointer is a pointer to a C structure of type “USER_DATA”, which contains a member called “ud_handle_resp”, which is a function pointer to handle each of the response attributes in turn. There are many other ways a response callback function could be written. This program assumes the existence of the following functions:

panic — This function is executed when an error occurs.

```
#include <stdio.h>
#include “dhs.h”

typedef struct
{
       int           t;
       DHS_CB_FN_PTR ud_handle_resp;
} USER_DATA;

void   my_response_callback
(
       DHS_CONNECT   connect,      /* Connection the command was sent to.           */
       DHS_TAG       tag,          /* Tag that identifies the command.              */
       char          *command,     /* The command string.                           */
       DHS_CMD_STATUScmdStatus,    /* The current command status.                   */
       char          *string,      /* Description of command status.                */
       DHS_AV_LIST   avList,       /* Command response AV list.                     */
       void          *userData     /* The commands user data pointer.               */
)
{
       int           i;
       DHS_DATA_TYPE type;         /* Type of attribute being examined.             */
       int           ndims;        /* Number of dimensions of the attribute.        */
       unsigned long dims[7];      /* Size of each dimension.                       */
       void          *value;       /* Pointer to the attribute value.               */
       char          *name;        /* The attribute name.                           */
       DHS_STATUS    status;       /* DHS library status value.                     */
       DHS_AV_ID     attrib;       /* Attribute id.                                 */


       status = DHS_S_SUCCESS;


       /*
        * Handle each of the attributes in the response.
        */

       for ( i = 0;; i++ )
       {
              /*
               * Get the information about the attribute.
               */

              attrib = dhsAvIndex( avList, i, &status );
              dhsAvInfo( attrib, &name, &type, &ndims, dims, &value, &status );
              if ( status != DHS_S_SUCCESS )
              {
                     status = DHS_S_SUCCESS;
                     break;
              }


              /*
               * Use the ud_handle_resp member of the USER_DATA structure to handle the
               * attribute.
               */

              ((USER_DATA *) userData)->ud_handle_resp( type, ndims, value );
       }


       /*
        * Check to see if the command is done.
        */

       switch ( cmdStatus )
       {
              case DHS_CS_IDLE:
              case DHS_CS_BUSY:
                     break;


              /*
               * All of the following command states are terminal states, so the command
               * tag should be freed.
```

```
         */

        case DHS_CS_ERROR:
               (void) fprintf( stderr, “command execution error.\n %s\n”, string );
               dhsTagFree( tag, &status );
               break;
        case DHS_CS_ABORTED:
               (void) fprintf( stderr, “command execution aborted.\n %s\n”, string);
               dhsTagFree( tag, &status );
               break;
        case DHS_CS_DONE:
               (void) fprintf( stderr, “command execution complete.\n %s\n”, string);
               dhsTagFree( tag, &status );
               break;
    }


    /*
     * Check the status value.
     */

    if ( status != DHS_S_SUCCESS )
    {
        panic();
    }
}
```

7.4 Error callback function

Following is an example of an error callback function. This function could be used in either a client or a server.

```
#include <stdio.h>
#include “dhs.h”

void    my_error_callback
(
        DHS_CONNECT    connect,      /* Connection causing the error.               */
        DHS_STATUS     errorNum,     /* Error number.                               */
        DHS_ERR_LEVEL  errorLev,     /* Error level.                                */
        char           *msg,         /* Pointer to the error message string.        */
        DHS_TAG        tag,          /* Tag of the command with the error.          */
        void           *userData     /* Command’s user data pointer.                */
)
{
        switch ( errorLev )
        {
               case DHS_EL_SEVERE:
                      (void) fprintf( stderr, “Severe error: %d\n%s\n”, errorNum, msg );

                      /*
                       * The DHS library is no longer safe to use, so exit the program.
                       */

                      exit( -1 );
                      break;
               case DHS_EL_ERROR:
                      (void) fprintf( stderr, “Error, operation failed: %d\n%s\n”,
                                     errorNum, msg );
                      break;
               case DHS_EL_WARNING:
                      (void) fprintf( stderr, “Warning: %d\n%s\n”, errorNum, msg );
                      break;
               case DHS_EL_INFO:
                      (void) fprintf( stderr, “Note: %d\n%s\n”, errorNum, msg );
                      break;
        }
}
```

7.5 Client aborting a command on a server

This example assumes the command server supports an “abort” command. This example waits for the abort command action to complete, and explicitly checks the status, but this could be done through callback routines. This program assumes the existence of the following functions:

panic — This function is executed when an error occurs.

```
#include <stdio.h>
#include “dhs.h”

abortCmd
(
       DHS_CONNECT   dhsServer,    /* The server the command was sent to.            */
       DHS_TAG       dhsTag        /* The tag of the abort command.                  */
)
{
       DHS_TAG       targetTag;    /* The tag of the command to be aborted.          */
       DHS_AV_LIST   avList;       /* The AV list of the abort command.              */
       char          *msg;         /* String describing the command status.          */
       DHS_STATUS    status;       /* DHS library return status.                     */


       status = DHS_S_SUCCESS;


       /*
        * Create a new attribute value list.
        */

       avList = dhsAvListNew( &status );


       /*
        * Add the tag of the command to be aborted to the AV list.
        */

       dhsAvAdd( avList, “targetTag”, DHS_DT_TAG, 0, NULL, targetTag, &status );


       /*
        * Send the command.
        */

       dhsTag = dhsApply( dhsServer, “abort”, avList, NULL, &status );


       /*
        * Wait for the abort command to complete.
        */

       dhsWait( 1, &dhsTag, &status );


       /*
        * Check the status of the abort command.
        */

       if ( dhsStatus( dhsTag, &msg, &status ) != DHS_CS_DONE )
       {
              (void) fprintf( stderr, “Failed to abort command.\n” );
              if ( msg != NULL )
              {
                     (void) fprintf( stderr, “%s\n”, msg);
              }
       }


       /*
        * Free the command tag.
        */

       dhsTagFree( dhsTag, &status );


       /*
```

```
         * Free the attribute value list.
         */

        dhsAvListFree( avList, &status );


        /*
         * Check the DHS library status.
         */

        if ( status != DHS_S_SUCCESS )
        {
               panic();
        }
}
```

7.6 Initialization of a server application

The following sample code initializes the DHS library and sets up the error and command call-back functions. This program assumes the existence of the following functions:

panic — This function is executed when an error occurs.

```
#include “dhs.h”

main
(
        int    argc,
        char   **argv
)
{
        void          my_error_callback();       /* Error callback function.        */
        void          my_command_callback();     /* Command callback function.      */
        DHS_STATUS    status;                    /* DHS library status.             */


        status = DHS_S_SUCCESS;


        /*
         * Initialize the DHS library with space for 1Mb of messages.
         */

        dhsInit( “servers_name”, 20, &status );


        /*
         * Set up the callback functions.
         */

        dhsCallbackSet( DHS_CBT_ERROR, my_error_callback, &status );
        dhsCallbackSet( DHS_CBT_COMMAND, my_command_callback, &status );


        /*
         * Start the event loop.
         */

        dhsEventLoop( DHS_ELT_WAIT, &status );


        /*
         * When the event loop returns, we must be done.
         */

        dhsExit( &status );


        /*
         * Check the DHS library status.
         */

        if ( status != DHS_S_SUCCESS )
        {
```

```
        panic();
    }

    exit( 0 );
}
```

7.7 Server command callback function

In real life the following function would be much more modular. This program assumes the existence of the functions:

panic — Executed to shut down the program when something bad happens.

THREAD_CREATE — Execute a function in a new thread.

command_abort — This function causes the processing associated with the execution of another command to be stopped.

verify_command — Verify that the parameters required by a command are present and correct.

```
#include “dhs.h”

void   my_command_callback
(
       DHS_CONNECT   connect,      /* in: Connection the command was sent to.       */
       DHS_TAG       tag,          /* in: Tag that identifies the command.          */
       char          *command,     /* in: The command string.                       */
       DHS_AV_LIST   avList        /* in: Command response AV list.                 */
)
{
       DHS_AV_ID     attrib;
       DHS_DATA_TYPE type;         /* Type of the attribute.                        */
       int           ndims;        /* Number of dimensions of the attribute.        */
       unsigned long dims[7];      /* Size of each attribute dimension.             */
       char          *name;        /* Name of the attribute.                        */
       DHS_TAG       targetTag;    /* Tag of the command to abort.                  */
       DHS_AV_LIST   response;     /* Response to send to the client.               */
       DHS_STATUS    status;       /* DHS library status.                           */


       status = DHS_S_SUCCESS;


       /*
        * Verify the command. The client system is waiting for the first response before
        * it continues, so the next section of code should happen quickly.
        */

       if ( verify_command( command, avList ) == OK )
       {
              dhsCmdResponse( connect, tag, DHS_CS_BUSY, “Command busy”, NULL, &status );
       }
       else
       {
              dhsCmdResponse( connect, tag, DHS_CS_ERROR,
                            “Command failed verification”, NULL, &status );
       }


       /*
        * Figure out which command was issued.
        */

       if ( strcmp( command, “abort” ) == 0 )
       {
              /*
               * Command to abort a command, get the target command tag from the AV list.
               */

              attrib = dhsAvFind( avList, “targetTag”, &status );
              dhsAvInfo( attrib, &name, &type, &ndims, dims, &targetTag, &status );
```

```
                /*
                 * Abort the command in a separate thread. (It may not be necessary to
                 * do this in a thread if the command_abort function can be guaranteed to
                 * execute quickly). This assumes the abort always succeeds, which may not
                 * be true in real life.
                 */

                THREAD_CREATE( command_abort( targetTag ) );
                dhsCmdResponse( connect, tag, DHS_CS_DONE, “Abort command done”, NULL,
                              &status );
        }
        else if ( strcmp( command, “getTime” ) == 0 )
        {
                /*
                 * Command to return the current time to the calling system.
                 */

                response = dhsAvListNew( &status );
                dhsAvAdd( response, “currentTime”, DHS_DT_INT32, 0, NULL, time( NULL ),
                              &status );
                dhsCmdResponse( connect, tag, DHS_CS_DONE, “Command action is complete”,
                              response, &status );
                dhsAvListFree( response, &status );
        }


        /*
         * Check the DHS library status.
         */

        if ( status != DHS_S_SUCCESS )
        {
                panic();
        }
}
```

8.0 Debugging

The DHS work package will provide a Command Server emulator (program **dhsServer**) which will run stand alone on a Sun Sparcstation running Solaris 2.X system and emulate the server functions described in the document. The emulator will be used by the other Gemini systems to test and debug their DHS interface. The emulator will receive commands from client systems, display the command and attribute value pairs included with the command.

The DHS library provides a **dhsDebugLevel** function which allows application programmers to change the amount of information provided by the DHS library. All debugging messages are delivered to the application via the error callback function.

The IMP library underlying the DHS library has a set of routines to allow low level debugging of IMP processes. These routines include **impdump**, **impbindump**, and **tasklog**. These routines are not for the faint of heart.

9.0 Logging and Alarms

The DHS library will **not** log any activity using the logging system described in [9]. Any required logging should be done by the application using the library.

10.0 Development and Test Factors

10.1 Project Control

The project has control over the standard set of functions and data structures used for this interface. New functions and data structures should only be added after consultation with the project.

Any new functions or data structures should be added to this document after a change control process which involves a period of time for general comment.

The management of this document, and any arbitration between the various work packages will be carried out by the Gemini project office.

10.2 Deliverables

The Data Handling System work packages will deliver the DHS library as described in this document.

The Data Handling System work package will deliver a Command Server emulator as described in Section 8.0 on page 41.

The Data Handling work package will provide documentation for users of the DHS library and documentation for maintenance of the DHS library.