

## Experion HS Application Development Guide

EHDOC-XXX5-en-500A

April 2017

Release 500

---

## **Disclaimer**

This document contains Honeywell proprietary information. Information contained herein is to be used solely for the purpose submitted, and no part of this document or its contents shall be reproduced, published, or disclosed to a third party without the express permission of Honeywell International Sàrl.

While this information is presented in good faith and believed to be accurate, Honeywell disclaims the implied warranties of merchantability and fitness for a purpose and makes no express warranties except as may be stated in its written agreement with and for its customer.

In no event is Honeywell liable to anyone for any direct, special, or consequential damages. The information and specifications in this document are subject to change without notice.

Copyright 2017 - Honeywell International Sàrl

---

# Contents

<b>Contents</b> .....	<b>3</b>
<b>About this guide</b> .....	<b>12</b>
<b>Server API reference</b> .....	<b>13</b>
Prerequisites .....	13
<b>About the Server API development environment</b> .....	<b>15</b>
Using Microsoft Visual Studio to develop Server API applications .....	15
Compiling and linking C and C++ projects .....	15
Setting up debugging utilities and tasks .....	15
Disabling the default debugger (Dr Watson) .....	16
Folder structures for C/C++ applications .....	16
Multithreading considerations for Server API applications .....	18
Error codes in the Server API .....	19
Validating IEEE 754 special values .....	20
<b>Implementing a Server API application</b> .....	<b>23</b>
Application choices .....	23
Programming languages .....	23
Type of application .....	23
C/C++ application template .....	23
Definitions .....	24
Initialization .....	24
Main body of a utility .....	25
Main body of a task .....	25
Data types .....	26
Writing messages to the log file .....	28
Server redundancy .....	29
Developing an OPC client .....	29
Developing an ODBC client .....	29
<b>Controlling the execution of a Server API application</b> .....	<b>30</b>
Starting an application .....	30
Running a utility from the command line .....	30
Selecting an LRN for a task .....	30

Starting a task automatically .....	30
Starting a task manually .....	31
Activating a task .....	32
Activating a task on a regular basis .....	32
Activating a task while a point is on scan .....	33
Activating a task when a status point changes state .....	34
Activating a task when a Station function key is pressed .....	34
Activating a task when a Station menu item is selected .....	35
Activating a task when a display button is selected .....	35
Activating a task when a Station prompt is answered .....	35
Activating a task when a display is called up .....	35
Activating a task when a report is requested .....	35
Activating a task from another task .....	36
Testing the status of a task .....	37
Monitoring the activity of a task .....	37
<b>Accessing server data .....</b>	<b>38</b>
Introduction to databases .....	38
The server database .....	38
Physical structure .....	39
Logical structure .....	40
Flat logical files .....	40
Object-based real-time database files .....	43
Strings in the logical file structure .....	43
Ensuring database consistency .....	43
Accessing acquired data .....	44
Identifying a point .....	44
Identifying a parameter .....	44
Accessing parameter values .....	45
Using point lists .....	45
Controlling when data is acquired and processed for standard points .....	46
Accessing process history .....	47
Accessing blocks of history .....	47
Accessing other data .....	48
Accessing logical files .....	48
Accessing memory-resident files .....	49

DIRTRY (The first logical file in the server database) .....	49
Accessing user-defined data .....	50
Displaying and modifying user table data .....	50
Setting up user tables using the UTBBLD utility .....	50
UTBBLD usage notes .....	57
Using the database scanning software .....	58
<b>Working from a Station .....</b>	<b>59</b>
Running a task from a Station .....	59
Routine for generating an alarm .....	59
Routine for using the Station Message and Command Zones .....	59
Routine for printing to a Station printer .....	60
<b>Developing user scan tasks .....</b>	<b>61</b>
Designing the database for efficient scanning .....	62
Example user scan task .....	62
C/C++ version .....	62
<b>Development utilities .....</b>	<b>74</b>
ADDTSK .....	74
CT .....	74
databld .....	75
DBG .....	75
DT .....	76
ETR .....	76
FILDMP .....	77
FILEIO .....	78
REMTSK .....	79
TAGLOG .....	80
USRLRN .....	81
<b>Application Library for C and C++ .....</b>	<b>82</b>
c_almmmsg_...() .....	82
AssignLrn() .....	84
c_chrint() .....	85
ctofstr() .....	86
c_dataio_...() .....	87
DeassignLrn() .....	96

c_deltask()	97
DbletoPV()	98
dsply_lrn()	100
c_ex()	100
ftocstr()	101
c_gbload()	102
c_gdbcnt()	103
c_getapp()	104
GetGDAERRcode()	105
c_geterrno()	106
c_gethstpar_..._2()	107
c_getlrn()	111
c_getlst()	112
c_getprm()	113
c_getreq()	116
c_givlst()	118
hsc_asset_get_ancestors()	119
hsc_asset_get_children()	120
hsc_asset_get_descendents()	122
hsc_asset_get_parents()	123
hsc_em_FreePointList()	124
hsc_em_GetLastPointChangeTime()	125
hsc_em_GetRootAlarmGroups()	125
hsc_em_GetRootAssets()	126
hsc_em_GetRootEntities()	128
hsc_enumlist_destroy()	129
hsc_GUIDFromString()	130
hsc_insert_attrib()	131
Attribute Names and Index Values	135
Valid Attributes for a Category	138
hsc_insert_attrib_byindex()	138
hsc_IsError()	142
hsc_IsWarning()	143
hsc_lock_file()	144
hsc_lock_record()	145

hsc_notif_send()	146
hsc_param_enum_list_create()	150
hsc_param_enum_ordinal()	152
hsc_param_enum_string()	154
hsc_param_format()	154
hsc_param_limits()	156
hsc_param_subscribe()	158
hsc_param_list_create()	159
hsc_param_name()	161
hsc_param_number()	162
hsc_param_range()	163
hsc_param_type()	165
hsc_param_value()	167
hsc_param_value_of_type()	169
hsc_param_values()	170
hsc_param_value_put()	174
hsc_param_values_put()	176
hsc_param_value_save()	178
hsc_pnttyp_list_create()	180
hsc_pnttyp_name()	182
hsc_pnttyp_number()	184
hsc_point_entityname()	185
hsc_point_fullname()	186
hsc_point_get_children()	186
hsc_point_get_containment_ancestors()	188
hsc_point_get_containment_children()	189
hsc_point_get_containment_descendents()	190
hsc_point_get_containment_parents()	191
hsc_point_get_parents()	193
hsc_point_get_references()	194
hsc_point_get_referers()	195
hsc_point_guid()	196
hsc_point_name()	197
hsc_point_number()	198
hsc_point_type()	199

hsc_StringFromGUID() .....	201
hsc_unlock_file() .....	202
hsc_unlock_record() .....	203
HsctimeToDate() .....	204
HsctimeToFiletime() .....	204
infdouble() .....	205
inffloat() .....	206
Int2toPV() .....	206
Int4toPV() .....	208
c_intchr() .....	209
IsGDAerror() .....	210
IsGDAnoerror() .....	211
IsGDAwarning() .....	212
isinfdouble() .....	212
isinffloat() .....	213
isnandouble() .....	214
isnanfloat() .....	215
Julian/Gregorian date conversion() .....	216
c_logmsg() .....	217
c_mzero() .....	218
nandouble() .....	219
nanfloat() .....	219
c_oprstr_...() .....	220
c_pps_2() .....	223
c_ppsw_2() .....	225
c_ppv_2() .....	226
c_ppvw_2() .....	227
PritoPV() .....	228
c_prsend_...() .....	229
RealtoPV() .....	231
c_rqtskb...() .....	232
c_sps_2() .....	235
c_spsw_2() .....	236
c_spv_2() .....	237
c_spvw_2() .....	238

c_stcupd()	239
stn_num()	240
StrtoPV()	240
TimetoPV()	242
c_tmstop()	243
c_tmstrt_...()	244
c_trm04()	245
c_trmstk()	246
c_tstskb()	247
c_upper()	248
c_wdon()	249
c_wdstrt()	249
c_wttskb()	250
Backward-compatible functions	251
c_badpar()	252
c_gethstpar_...()	253
c_pps()	257
c_ppsw()	258
c_ppv()	260
c_ppvw()	261
c_sps()	263
c_spsw()	264
c_spv()	265
c_spvw()	267
Examples	268
<b>Network API reference</b>	<b>270</b>
Prerequisites	270
<b>Network application programming</b>	<b>271</b>
About the Network API	271
Specifying a network server in a redundant system	272
Summary of Network API functions	272
Using the Network API	274
Determining point numbers	274
Determining parameter numbers	275

Accessing point parameters .....	276
Accessing historical information .....	277
Accessing user table data .....	279
Looking up error strings .....	284
Functions for accessing parameter values by name .....	285
Using Microsoft Visual Studio or Visual Basic to develop Network API applications .....	286
Using the Visual Basic development environment .....	286
Changing packing settings when compiling C++ applications .....	287
Folder structures for C/C++ applications .....	287
Network API applications fail to run .....	288
<b>Network API Function Reference .....</b>	<b>290</b>
Functions .....	290
hsc_bad_value .....	290
hsc_napierrstr .....	291
rgetdat .....	291
rhsc_notifications .....	294
rhsc_param_hist_date_bynames .....	298
rhsc_param_hist_offset_bynames .....	298
rhsc_param_hist_dates_2 .....	306
rhsc_param_hist_offsets_2 .....	306
rhsc_param_numbers_2 .....	311
rhsc_param_value_bynames .....	314
rhsc_param_value_put_bynames .....	318
rhsc_param_value_put_sec_bynames .....	323
rhsc_param_value_puts_2 .....	325
rhsc_param_values_2 .....	330
rhsc_point_numbers_2 .....	336
rputdat .....	338
Backward-compatibility Functions .....	341
hsc_napierrstr .....	341
rgethstpar_date .....	342
rgethstpar_ofst .....	342
rgetpnt .....	345
rgetval_numb .....	347
rgetval_ascii .....	347

rgetval_hist .....	347
rgetpntval .....	350
rgetpntval_ascii .....	350
rhsc_param_hist_dates .....	351
rhsc_param_hist_offsets .....	352
rhsc_param_value_puts .....	355
rhsc_param_values .....	360
rhsc_param_numbers .....	366
rhsc_point_numbers .....	368
rputpntval .....	371
rputpntval_ascii .....	371
rputval_hist .....	372
rputval_numb .....	372
rputval_ascii .....	372
Diagnostics for Network API functions .....	375
<b>Using Experion's Automation Objects .....</b>	<b>380</b>
Server Automation Object Model .....	380
HMIWeb Object Model .....	380
Station Scripting Objects .....	381
Creating an SSO .....	382
Registering an SSO .....	383
Implementing SetStation Object .....	383
Implementing a Detach method .....	384
Station Object Model .....	384
<b>Glossary .....</b>	<b>386</b>
<b>Notices .....</b>	<b>404</b>

---

## About this guide

This guide describes how to write applications for Experion, Release 500.

### Revision history

Revision	Date	Description
A	April 2017	Initial release of document.

### How to use this guide

To write applications using	Go to
Server API	<i>Server API reference</i> on the next page
Network API	<i>Network API reference</i> on page 270
Object Models	<i>Using Experion's Automation Objects</i> on page 380

---

## Server API reference

This section describes how to write applications for Experion using the Server API.

---

### Attention:

An application written for the local server is only available locally to the server, and not remotely across a network. For information about writing applications that can access the server database across a network, see *Network application programming* on page 271.

---

For:	Go to:
Prerequisites	<i>Prerequisites</i> below
An introduction to the development environment	<i>About the Server API development environment</i> on page 15
An introduction to the types of applications you can develop: <i>tasks</i> and <i>utilities</i>	<i>Implementing a Server API application</i> on page 23
Information about integrating your application with Experion	<i>Controlling the execution of a Server API application</i> on page 30
A description of the Experion database, and how you access data	<i>Accessing server data</i> on page 38
Information about writing applications for Station	<i>Working from a Station</i> on page 59
Information about writing interfaces for unsupported controllers ( <i>user scan tasks</i> )	<i>Developing user scan tasks</i> on page 61
Development utilities	Development Utilities
C application library	<i>Application Library for C and C++</i> on page 82

### Prerequisites

Before writing applications for Experion, you need to:

- Install Experion and third-party software as described in the *Getting Started with Experion Software Guide*.
- Be familiar with user access and file management as described in the *Server and Client Configuration Guide*.

## Prerequisite skills

This guide assumes that you are an experienced programmer with a good understanding of either C or C++.

---

**Attention:**

An application written for the local server using the Server API must be written in C or C++, as the Server API does not support other programming languages such as Visual Basic or the .NET languages

---

It also assumes that you are familiar with the Microsoft Windows development environment and know how to edit, compile and link applications.

---

## About the Server API development environment

If development is to be conducted on a target system, consideration must be given to the potential impact on the systems performance during the development.

Use of these facilities requires a high level of expertise in the development tools, including C compiler, C++ compiler, linker, make files and batch files.

### Using Microsoft Visual Studio to develop Server API applications

You use Microsoft Visual Studio to develop Server API applications.

#### Compiling and linking C and C++ projects

To compile and link your project in Microsoft Visual Studio, select **BuildBuild** <project name>.

---

#### Attention:

This procedure will only work with C and C++ projects. After compiling and linking, all executable files should be copied to the **run** folder.

---

### Changing packing settings when compiling applications

When using C++ in Visual Studio, certain settings that affect the interpretation of header files should *not* be changed from their defaults when compiling applications, because it will cause the Experion header files to be interpreted incorrectly.

If you do need to change the packing setting, use **#pragma** lines instead to change the settings for your code but not for the Experion headers. For example, the following code is legitimate:

```
#include <Experion header>
#pragma pack(push, 2)
#include <Customer Code>
#pragma pop()
#include <More Experion headers>
```

#### Setting up debugging utilities and tasks

Before a utility or task can be debugged, it needs to be compiled and linked with debugging information.

To compile and link with debugging information for C/C++ applications using Visual Studio, select the Debug build as the active configuration. To do this, select **Build > Configuration Manager** and then select the debug build.

### To set up the debugging utilities:

1. Open the project using Visual Studio.
2. Open up the source files for the utility. In the case of a C/C++ program, the filenames will not have been altered.
3. Set break points as required in the source files.
4. Start the debugger (select **DebugGo**).

### To set up the debugging tasks:

1. Run the server utility program **DBG**, passing it the LRN the task is using.
2. Complete the procedure described for debugging a utility.
3. Execute and ETR on the LRN.
4. Debug the program.

---

#### Attention:

When using the **DBG** utility, make sure that no other application executes a `gbload()` before your application, otherwise it will be assigned the LRN you specified in step *Run the server utility program DBG, passing it the LRN the task is using.* above.

---

### Disabling the default debugger (Dr Watson)

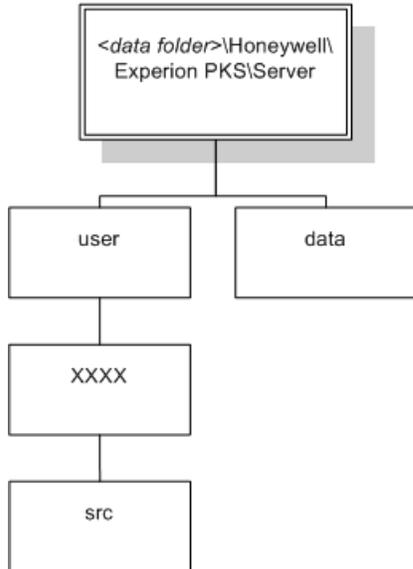
By default, every time the server starts it sets Dr Watson as the default debugger.

You can prevent the server doing this (which allows you to use another debugger such as Visual Studio) by updating the registry as follows: go to the registry key **HKEY\_LOCAL\_MACHINE\Software\Wow6432Node\Honeywell**, and create a string value called **EnableDebug**. (You do not have to specify a value.)

### Folder structures for C/C++ applications

The folder structures shown below should be used for development of C/C++ Server API applications which will run on the server.

**<data folder>** server folder structure

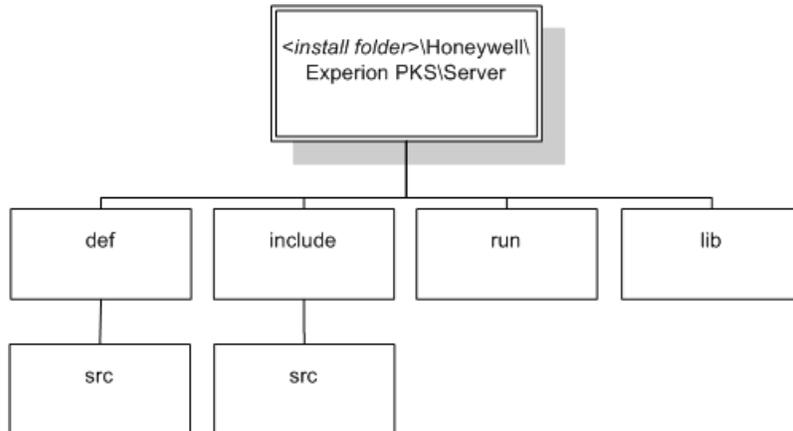


**<data folder>** is the location where Experion data is stored. For default installations, this is **C:\ProgramData**.

The **user/XXXX/src** folder contains all source code files and make files for a particular application. **XXXX** should be representative of the function of the application.

The **server/data** folder contains all server database files.

**<install folder>** server folder structure



**<install folder>** is the location where Experion is installed. For default installations this is **C:\Program Files (x86)**.

The **def** folder contains real time database definitions.

The **include** and **include/src** folders contain global server definitions, such as system constants or data arrays in the form of C/C++ include files.

The **run** folder contains all server programs (including applications). This folder is included in the path of any server user.

The **lib** folder contains libraries used for application development.

## Multithreading considerations for Server API applications

Multithreading is a form of multitasking which allows an application to multitask within itself.

It is possible to write a multithreaded application that uses the application programming library but it is not recommended.

If there is a requirement for multithreading then the following should be observed:

- Call **c\_gbload** before any threads are created.
- Keep all access to the application programming library serialized. This can be achieved in two ways. Keep all calls to the application programming library in one thread (for example, the main thread of the program) or encase any calls in a critical section. See the code fragment below for an example.

If a multithreaded task has been created with an LRN, only the main thread of the task is associated with that LRN. All other threads will need to obtain their own LRN if they need one. Threads cannot share an LRN. Note that a free LRN can be obtained by using the **AssignLrn()** function (using **-1** as the parameter), and the **DeassignLrn()** function should be used when the thread terminates.

---

### Example

```
// Main code segment
CRITICAL_SECTION serAPI;
/*critical section for calling the Server APIs */
InitializeCriticalSection(&serAPI);
if (c_gbload())
{
    /* Could not attach to database */
    exit(-1);
}
... Create threads
... Execution continues on
//End of main code segment

// Thread code segment
EnterCriticalSection(&serAPI);
```

---

```
... Call to Experion API
LeaveCriticalSection(&serAPI);
```

---

## Error codes in the Server API

Error status information is returned from functions in the Server API using one of two different methods:

- The first method is used by most of the functions in the Server API, which return **FALSE (0)** if they completed successfully, otherwise return **TRUE (-1)**.
- The second method is to return the error status directly as the result of the function, where otherwise, **FALSE (0)** is returned if the function completed successfully.

If **TRUE (-1)** is returned, the error code will be set to the return status of the function. This value can be checked to see whether it indicates an error or a warning using the functions **hsc\_IsError()** and **hsc\_IsWarning()**.

To safely retrieve the value of the error code, call the function **c\_geterrno()** immediately after the function return (before it gets overwritten).

---

### Attention:

Any applications that use the function **c\_geterrno()** must include the **M4\_err.h** header file, that is, **#include <src/M4\_err.h>**.

---

### Example

This example shows how to check the error status:

```
if (c_server_api_function(arg1, arg2) == -1)
{
int errcode = c_geterrno();
if (hsc_IsError(errcode))
{
// an error has been issued
// handle error here
// or pass to your error handler
```

---

```
}  
else  
{  
  // a warning has been issued  
  // handle warning here  
  // or may be safe to ignore  
}  
}
```

---

## Validating IEEE 754 special values

These functions assist in validating IEEE special values INF (Infinity) and NaN (Not A Number) used when communicating with a Controller.

- *infdouble()* on page 205
- *inffloat()* on page 206
- *isinfdouble()* on page 212
- *isinffloat()* on page 213
- *nandouble()* on page 219
- *nanfloat()* on page 219
- *isnandouble()* on page 214
- *isnanfloat()* on page 215

---

### Attention:

These functions are platform dependent (only valid for an INTEL X86 system).

---

## IEEE 754 Standard

The IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985), is the most widely-used standard for floating-point computation. It defines formats for representing floating-point numbers (including negative zero and denormal numbers) and special values

(infinities and NaNs) together with a set of floating-point operations that operate on these values.

## IEEE 754 floating point binary format

In IEEE 754, binary floating-point numbers are stored in a sign-magnitude form where the most significant bit (MSB) is the sign bit (positioned on the left in *The IEEE 754 floating point binary format* below), exponent is the biased exponent, and the mantissa or 'fraction' is the significand without the most significant bit.

*The IEEE 754 floating point binary format*



Element	Description
1	Sign
2	Exponent
3	Mantissa

The LSB (Least Significant Bit, the one that if changed would cause the smallest variation of the represented value) with index 0 is positioned on the right.

## IEEE data format description

(Intel 80387) IEEE compliant single precision format which is 32 bits in size containing 1 sign bit, 8 bit exponent, 23 bit mantissa. (Intel 80387) IEEE compliant double precision format is similar but is 64 bits in size containing 1 sign bit, 11 exponent bits, 52 mantissa bits.

## IEEE 754 special values

In IEEE 754, Exponent field values of all 0s and all 1s are used to denote special values.

### Zero

The value **Zero (0)** is represented with an exponent field of zero and a mantissa field of zero. Depending on the sign bit, it can be a positive zero or a negative zero. Thus, **-0** and **+0** are distinct values, though they are treated as equal.

### Infinity

The value infinity is represented with an exponent of all 1s and a mantissa of all 0s. Depending on the sign bit, it can be a positive infinity or negative infinity. The infinity is used in case of the saturation on maximum representable number so that the computation could continue.

## NaN

The value **NaN** (Not a Number) is represented with an exponent of all 1s and a non-zero mantissa. NaN's are used to represent a value that does not represent a real number, and are designed for use in computations that may generate undefined results, so that the computations can continue without a numeric value. The **NaN** value usually propagates to the result, to help indicate that a numeric value was missing in the calculation.

There are two categories of **NaN**:

- **QNaN** (Quiet **NaN**) is a **NaN** with the most significant fraction bit set (denotes indeterminate operations).
- **SNaN** (Signalling **NaN**) is a **NaN** with the most significant fraction bit clear (denotes invalid operations).

## See also

*Data types* on page 26

---

# Implementing a Server API application

## Application choices

Before you can implement a Server API application you will need to make a choice on the programming language you will use and the type of application you are going to implement.

## Programming languages

The Server Application Programming Library only supports the development of C/C++ applications.

The language you choose to use will largely depend on your experience with these languages. Alternatively, it is possible to develop an OPC Client to access server data via the Experion OPC Server. The client can be written in C or C/++.

## Type of application

There are two types of application you can develop:

- **Utility.** A utility runs interactively from the command line using the Experion Command Prompt.

Utilities typically perform an administrative function or a function that is performed occasionally.

A utility can prompt the user for more information and can display information directly to the user via the command prompt window.

An example of a utility is an application to dump the contents of a database file to the command prompt window, for example *FILDMP* on page 77.

- **Task.** Tasks are usually dormant, waiting for a request to perform some form of function. When they are activated, they perform the function and then go back to sleep to wait for the next request to come along.

An example of a task is an application that periodically fetches some point values, performs a calculation on the values and stores the result back in the database.

## C/C++ application template

This section provides a generic C/C++ application template that can be used for any application you may develop. Again, it doesn't contain much functionality but it should give you an idea of the parts necessary for an application.

## Definitions

A C/C++ application also needs to contain several include files that declare and define items used by the application programming library routines. These include files should be incorporated in the main source file as well as any function source files that make calls to the application programming library routines. The include files used by this template are:

Include file	Description
<b>defs.h</b>	Defines system constants and some useful macros.
<b>M4_err.h</b>	Defines error code constants.
<b>files</b>	Defines all the logical file numbers of the database.
<b>parameters</b>	Defines all the point parameters of a point.
<b>GBtrbtbl.h</b>	Defines the structure of one of the database files.

The include folder also contains many other **GBxxxxxx.h** include files that may be needed if you make calls to other application programming library routines.

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <files>
#include <parameters>
#include <src/GBtrbtbl.h>
char *progrname='myapp.c';

main()
{
uint2 ierr;
char string[80];
struct prm prmbk;
```

## Initialization

The first function you must call in any application is GBLOAD. This function makes the global common memory available to the application, allowing it to access the memory-resident part of the database. If this call fails you should terminate the program. This function should only be called once for the whole program.

```
if (ierr=c_gbload())
{
//The next line number is 137
c_logmsg(progrname, '137', 'Common
Load Error %d\n', ierr);
```

```
c_deltsk(-1);
c_trmtsk(ierr);
}
```

### Main body of a utility

This is where the majority of the work of the application is done. If your application requires arguments from the command line, you can call GETPAR to retrieve individual arguments. You can also use the **argv** and **argc** parameters if your utility has a C/C++ main function.

As the application is run interactively you can print messages to a Windows Command Prompt using the **printf** command and also scan messages back from the Command Prompt using the **scanf** command. A C++ application can use **std::cin** and **std::cout**.

After the application has completed its work you should call DELTSK and TRMTSK to mark the application for deletion and to terminate the application. It is your responsibility to close any files you may have opened in the application.

```
c_getpar(1,string,sizeof(string))
**** Perform some Function ****
printf('Results of Function\n');
c_deltsk(-1);
c_trmtsk(0);
}
```

### Main body of a task

The main body of a task is slightly different in that it is usually all contained in an endless loop. After the task is started, it will remain in this loop until the system is shut down.

As the application is not run interactively, you cannot 'scanf' responses from a Command Prompt. You can use the **c\_logmsg** function to write messages and information to the log file. The log file can be viewed by looking at the file **server\data\log.txt** with Notepad or the **tail** utility.

After the task enters the endless loop, it should call GETREQ to see if any other application has requested it to perform some function. The call to GETREQ will return a parameter block that provides information about who and why the task was requested. Based on this information, the task should perform the desired function, loop back up and check for the next request.

If no request is outstanding, the task should call TRM04 to cause it to go to sleep. This will cause the task to block (or hibernate) until the next request.

**When the next request comes along, the task:**

1. Returns from the TRM04 call.
2. Loops back up.
3. Gets the parameter block associated with the request.
4. Performs the function.
5. Continues.

```

while (1)
{
    if (c_getreq((int2 *) &prmbk))
    {
        int errcode = c_geterrno();
        if (errcode != M4_EOF_ERR)
        {
            /* it is a real error so report and */
            /* handle it */
        }
        /* Now terminate and wait for the */
        /* next request */
        c_trm04(ZERO_STATUS);
    }
    else
    {
        /* Perform some function */
    }
}

```

The `c_getreq` function will return a FALSE (0) if there is has been a request. It will return the error `M4_EOF_ERR` (0x21f) if there are no requests pending. If any other error is returned, then this should be reported and optionally handled.

**Data types**

In the definitions section, you may have noticed the use of the C/C++ data type `uint2`. This is one of several data types that are defined in the header file `defs.h`. This is necessary because different C and C++ compilers define the different sizes for: int, long, float, and double.

The following data types are used throughout the application programming library routines:

Data type	Description
int2	Equivalent to INTEGER*2

Data type	Description
uint2	Unsigned version of int2
int4	Equivalent to INTEGER*4
uint4	Unsigned version of int4

They are defined in the header file **defs.h**.

The **defs.h** file also provides several macros that should be used whenever the user wants to access int4, double or real database values, including user table values of these types. The available macros are:

Macro	Description
ldint4(int2_ptr)	Load an int4 value from the database (pointed to by int2_ptr).
stint4(int2_ptr, int4_val)	Store an int4 value in the database at the position pointed to by int2_ptr.
ldreal(int2_ptr)	Load a real value from the database.
streal(int2_ptr, real_val)	Store a real value in the database.
ldouble(int2_ptr)	Load a double value from the database.
stdouble(int2_ptr, dble_val)	Store a double value in the database.

These macros help to ensure that all types are properly assigned in C/C++ programs, to provide portability between different computer architectures.

One example of the use of such macros is provided below. This example shows how a C program would assign a floating point value to a variable, and also how a floating point value may be stored in the database.

```
#include <src/defs.h>
#include <src/GBsysflg.h>
.
float fval1;
float fval2;
.
.
.
/*load the seconds since midnight into the variable fval1 */
fval1 = ldreal(GBsysflg->syssec);
.
.
/* store the value of fval2 into the seconds since midnight */
streal (&GBsysflt->syssec, fval2);
```

.  
. .  
.

The other macros mentioned above are used in a similar manner. For the definitions of these macros, consult the **defs.h** file.

### Writing messages to the log file

When programming in C/C++ you should not use **printf** or **fprintf** calls, nor the **std::cout** or **std::cerr** streams to write messages to the log file. Instead, use the Server API routine **c\_logmsg()**.

It has the prototype:

```
void c_logmsg
(
    char*  progname,  //(in) name of program module
    char*  lineno,    //(in) line number in program module
    char*  format,    //(in) printf type format of message
    ...
);
```

Instead of:

```
printf('Point ABSTAT001 PV out of normal range (%d)\n' abpv);
```

OR

```
fprintf(stderr,'Point ABSTAT001 PV out of normal range (%d)\n',abpv);
```

OR

```
std::cout <<'Point ABSTAT001 PV out of normal range' <<abpv <<std::endl;
```

Use:

```
c_logmsg ('abproc.c', '134', 'Point ABSTAT001 PV out of normal range
(%d)', abpv);
```

---

#### Attention:

**c\_logmsg** handles all carriage control. There is no need to put line feed characters in calls to **c\_logmsg**.

---

If **c\_logmsg** is used to write messages in a utility, then the message will appear in the Command Prompt window.

## Server redundancy

If your task follows the guidelines described in this document and only accesses data from user tables and points, you do not have to do anything special for redundancy. (Your task doesn't need to determine which server is primary because GBLoad only allows the task on the primary to run.)

On a redundant system the task is started on both servers. If the server is primary, the task continues normal operation after GBLoad. However, if the server is backup the task waits at GBLoad.

When the backup becomes primary, the task continues on from GBLoad. In the meantime, what was the primary will reboot and restart as backup and the task will wait at GBLoad.

## Developing an OPC client

Experion provides an OPC Server which enables OPC clients to access Experion point data.

The Experion OPC Server supports two standard OPC interfaces—a custom interface for use by clients written in C, and an automation interface for use by clients written in Visual Basic. You can write an OPC client in either of these languages.

For more information about:

- The Experion OPC Server, see the topic, 'Accessing data from the Experion OPC Data Access Server.' in the *Server and Client Configuration Guide*
- OPC interfaces, see the OPC Standard. This standard can be downloaded from <http://www.opcfoundation.org>.

## Developing an ODBC client

Visual Basic or C++ applications can access the server database by using the Experion ODBC driver.

For more information about writing an application that uses the Experion ODBC driver, see the topic, 'Using the Experion ODBC driver with Visual Basic and C++' in the *Server and Client Configuration Guide*.

---

## Controlling the execution of a Server API application

### Starting an application

These topics describe how to start an application.

### Running a utility from the command line

After a utility has been compiled and linked, as described in *About the Server API development environment* on page 15, it is ready to be run from the command line. The utility's output should direct the user on what to do to use the utility.

### Selecting an LRN for a task

Before you start a task, you need to identify it within Experion by selecting a unique Logical Resource Number (LRN) for the task. The LRN range from 111 to 150 inclusive has been allocated to the user space for this purpose.

The **USRLRN** utility is provided to help you quickly identify a free user application LRN that can be allocated to your task. See the topic, 'usrlrn' in the *Configuration Guide*.

---

#### Attention:

All LRNs except for the user space (from 111 to 150 inclusive) are reserved by the server for internal use and should not be used for applications.

---

To use **USRLRN** and select one of the numbers it displays, at the Windows Command prompt, type:

#### **usrlrn**

When a task is executing, you can identify its LRN by calling the library routine **GETLRN**. This LRN is needed in some other library routines and it prevents you from having to hard-code it into your source code.

### Starting a task automatically

You can configure your system to start your task automatically whenever the server starts up. Your task will always be up and ready to be activated whenever the server system is running.

**Attention:**

Configuring the task to start automatically only takes effect after you have stopped and started the server. Also note that starting and activating a task are two separate activities. Once the task is started, it needs to be activated before any of its commands are executed.

**To configure your task to start automatically:**

1. Log on to Station with **MNGR** security level.
2. Choose **ConfigureApplication DevelopmentApplication Summary** to call up the Applications Summary display.
3. Click an empty record line to call up the **System Configuration Application** display.
4. Type a suitable descriptive title in **Description**.
5. Type the name of the executable without the **.exe** extension in **Task Name**. This is the name you use to link your application.
6. Type the LRN you have selected for your task in **Task LRN**. See *Selecting an LRN for a task* on the previous page.
7. Type **17** (the recommended priority for user tasks) in **Task Priority**.
8. The **Database Resources** options are used to store further configuration information about your application. The task may access this information by using the GETAPP function.

**Starting a task manually**

It can often be useful to start a task manually from the command line, either for debugging purposes or because you do not have the opportunity to stop and start the server to do it automatically. Several utilities are provided to allow you to manipulate a task from the command line.

The syntax for starting a task is:

```
addtsk namelrn [priority]
```

Item	Description
<b>name</b>	The executable file name of your task.
<b>lrn</b>	The LRN for the task, see <i>Selecting an LRN for a task</i> on the previous page.
<b>priority</b>	The priority of task execution (use <b>0</b> as a default).

To activate a task from the command line use:

```
etr lrn
```

To mark a task for deletion from a command line use:

```
remtsk lrn
```

where **lrn** is the task's LRN.

For details about these utilities, see *ADDTSK* on page 74, *ETR* on page 76 and *REMTSK* on page 79.

## Activating a task

After a task has been started it is ready to receive requests to be activated. The server can be configured to activate your task whenever one or more of the following events occurs.

When your task is woken from its TRM04 call by one of these events you can usually obtain more information about the event by calling GETREQ. The parameter block returned from GETREQ can provide event specific information that can be used to determine what action your task should take. Note that if GETREQ is not called, then the request will not be flushed from the request queue and no further requests to the task can be made.

The remainder of this section describes how to configure the server to activate your task for each of these events and also what event specific information you can obtain from the parameter block.

### Activating a task on a regular basis

To get the server to request your task on a regular basis you can make a call to the application programming library routine TMSTRT while the task is initializing. This will set up an entry in the server timer table that will cause the server to activate your task on a regular basis.

### To view the current timer table entries

1. In Station, choose **ConfigureApplication DevelopmentTask Timers** to call up the Task Timers display.

### To stop the periodic requests

1. To stop the periodic requests you can use **TMSTOP**. Note that the TMSTRT application programming library routine can also be used to activate your task once-off at some time in the future, rather than periodically.

When activated using this method, your task can call GETREQ to obtain the following information in the parameter block.

Word	Description
1	Set to 0.
2	param1 passed to TMSTRT.
3	param2 passed to TMSTRT.
4-10	Not used.

### Activating a task while a point is on scan

You may want to have an operator control when your task is to be requested on a regular basis. This can be done by using the PV Algorithm No 16: Cyclic Task Request.

While a point with this Algorithm is ON SCAN, it will cause the application task with the specified LRN to be activated on a regular basis. To configure the Algorithm in Quick Builder, you need to define the following parameters:

Parameter	Description
Block No.	Algorithm data block number. For details, see the topic, 'Algorithm blocks' in the <i>Configuration Guide</i> .
Task LRN	The logical resource number of your task.
Task Request Rate	The task request rate in seconds, (must be multiple of point scan rate).
Word 1(param1)	Must be a non-zero number.
Word 2-10 (param2-10)	Numerical parameters that will be passed to your task.

### Notes

- The algorithm block can also be configured from the Cyclic Task Request Algorithm display. Using the Point Detail display, click the Algorithm number to display the Algorithm configuration.
- This algorithm must be attached to either a Status or Analog point with no database or hardware address (that is, Controller number only).
- Time of the last request (in seconds) is stored by the system in ALG(04).

When activated using this method, your task can call GETREQ to obtain the following information in the parameter block:

Words 1 -10.

### Activating a task when a status point changes state

You may want to have a task requested based on some change in the field. This can be done by using the Action Algorithm 69: Status Change Task Request.

A single request is made to the task with the specified LRN each time the Status point changes to the nominated state (0–7). Alternatively, a nominated state of ALL (or –1) will request the task for all state transitions. To configure the Algorithm in Quick Builder, you need to define the following properties:

Property	Description
Block No.	The algorithm block used by this algorithm for this point. Each algorithm attached to each point should be assigned a unique block number. Use the <code>alglst</code> utility to find a free block.  See the topic titled "Algorithm blocks" in the <i>Server and Client Configuration Guide</i> for more information.
LRN of Task to Request	The Logical Resource Number of the task that is requested when the point changes to the specified state.  You can specify a system task or a custom task.
Task Request State	Select the state (0 to 7) that requests the task, or select ALL for all state transitions.
Parameter Block	The numerical parameter(s) passed to the task. Note that Word 1, Word 2, or Word 3 must be a non-zero number, otherwise the parameter block is not read and all other parameter values are ignored.

### Notes

- The algorithm block can also be configured from the Status Change Task Request Algorithm display. Using the Point Detail display, double-click the Action algorithm number to display the Algorithm configuration.
- This algorithm must be attached to a Status point.
- This algorithm does not queue requests to the task.

The task must call GETREQ to obtain the following information in the parameter block:

Words 1–10

### Activating a task when a Station function key is pressed

The Station function keys can be configured to activate a specific task. The function keys are configured for each Station. For details, see the topic 'Connection tab, Connection properties' in the *Server and Client Configuration Guide*.

### Activating a task when a Station menu item is selected

You can configure a menu item to activate your task. For details, see the topic 'Connection tab, Connection properties' in the *Server and Client Configuration Guide*.

### Activating a task when a display button is selected

If the operator only needs to activate your task when looking at a particular display, you can place a pushbutton object on that display. The pushbutton object is configured to activate your task. For details, see the *Display Building Guide* (for DSP displays) or the *HMIWeb Display Building Guide* (for HMIWeb displays).

### Activating a task when a Station prompt is answered

A task may often require information from an operator using a particular Station. You can prompt the operator to type a string in Station's Command Zone by using the OPRSTR routine. This routine displays a message prompt in the Message Zone and returns to the calling function.

When the operator has typed a response and pressed ENTER your task is re-activated, and you can call GETREQ to obtain the following information in the parameter block.

Word	Description
1	Parameter 1 passed to the OPRSTR routine.
2-10	Not used.

### Activating a task when a display is called up

You can develop an application task that sits behind a display and performs additional processing. The display can be configured to activate your task whenever it is called up, or at regular intervals while it is visible. For details, see the topic "Defining display and shape properties (DSP)" in the *Display Building Guide* (for DSP displays) or the topic "Display and shape properties (HMIWeb)" in the *HMIWeb Display Building Guide* (for HMIWeb displays).

### Activating a task when a report is requested

After a server report has been requested, you may require extra processing of the report in an application specific way. This is achieved by configuring the report to request your application task after the report generation is complete.

## To configure a report to activate a task

1. In Station, choose **ConfigureReports**.
2. Use the scroll bar to find the report you want to change and click the report name. The report details are displayed.
3. Click the **Definition** tab on the display. The report definition appears.
4. In the **Request program LRN** box, type the logical resource number of your task.

When activated using the display, your task can call GETREQ to obtain the following information in the parameter block.

Word	Description
1	Station number requesting the report
2	Not used
3	Report number
4–10	Not used

---

### Attention:

The report output file will reside in the **report** folder of the server and will have the name **RPTnnn** where **nnn** is the report number.

---

## Activating a task from another task

For a complicated application, you may need to implement a solution using more than one task. To synchronize the execution of each of your tasks, you can request one task from another.

Use the application programming library routine RQTSKB to request another task to be activated if it is not already active.

When activated using this method, the receiving task can call GETREQ to obtain the following information in the parameter block.

Word	Description
1-10	Values passed into the requesting tasks call to RQTSKB.

## Testing the status of a task

There are two library routines provided to allow you to wait for or check up on the status of another task.

In some cases you may want to suspend execution until another task has performed an operation for you. To do this, call the routine `WTTSKB` after you have activated the other task with `RQTSKB`. `WTTSKB` will block your task, and only return when the other task has called its own `TRM04`.

Rather than suspending your own task, you can check the status of the other task by calling the routine `TSTSKB`. This routine will indicate whether the specified task is active performing some function or dormant in a `TRM04` call.

## Monitoring the activity of a task

In some critical applications that you write, it may be desirable to know that the task written is actually working, and if not, to then take certain actions. Watchdog timers are provided for this purpose.

Watchdog timers are used to monitor tasks. They operate a countdown timer which is periodically checked for a zero or negative value. If the timer value is zero or negative, then the watchdog will trigger a certain predetermined action. The timer value can be reset at anytime by the task associated with that timer, thus avoiding the timeout condition.

Watchdog timers are started with a call to the watchdog start routine, `WDSTRT`, by the calling task. An action upon failure and a timeout interval (poll interval) must be specified. The following table describes the actions that can be taken on failure.

Action on failure setting	Description
<b>Alarm</b>	Generate an alarm upon failure.
<b>Reboot</b>	Restart the server system on failure.
<b>Restart</b>	Restart the task on first failure, and reboot the system on subsequent failures.

The tasks may then reset their watchdog timers by calling the watchdog timer pulse routine, `WDON`, which resets the countdown timer to the poll interval value.

For details on the routines, see `c_wdstrt()` on page 249 and `c_wdon()` on page 249 (C and C++).

### To check the watchdog timers:

1. In Station, choose **ConfigureApplication DevelopmentWatchdog Timers** to call up the Watchdog Timers display.

# Accessing server data

## Introduction to databases

Databases are a structured store of information to be referenced, altered or deleted at a later date. Many types of databases exist, but the majority of them can be classified into three main categories:

- **Relational.** Relational databases are used heavily in business applications where the data is represented as various tables, each containing a series of records and each record containing a set of fields. Due to the nature of the relational database structures, their strength lies in their ability to support ad hoc queries and quick searches.
- **Object oriented.** Object oriented databases are used more in Computer Aided Design (CAD) applications, where the data relationships are too complex to map into a table, record and field format. They are usually bound very closely to an object oriented language and provide better performance than relational databases.
- **Real-time.** Real-time databases are used in process control applications where the performance of the database is paramount. These databases usually consist of a memory-resident portion to ensure fast operation. The tasks that references the memory-resident fields can reference them just as if they were local variables in the program.

## The server database

A knowledge of the server database is essential for programming in the server environment. Use of the database involves considerations of both performance and maintenance to ensure minimal impact on other system functions. This section describes the internal structure of the server database to aid with this understanding.

The server system makes use of a real-time database to store its data. This data can be used throughout the whole server system, and by any applications that you intend to develop. The database provides the primary interface between an application and the standard server software.

The types of data stored in the server database can be classified as follows:

Type of data	Description
Acquired Data	Data that has been read from or is related to controllers.
Process History	A historical store of acquired data.
Alarms and Events	Details on alarm and event conditions that have occurred.
System Status	Details on the state of communications with remote devices.
Configuration Data	Details on how the server system has been configured to operate.

Type of data	Description
User Defined Data	Structures to store your own application specific data.

### Physical structure

The term *physical structure* of the server database is referring to the files that are used by the native operating system to store data. When using the application programming library routines you will only be referring to the logical structure of the database, but it is useful to understand how it is physically stored.

### The physical structure of the database

The database is made up of a number of files that reside in the **data** folder. The **data** folder is located in **<data folder>\Honeywell\Experion HS\Server**. Where **<data folder>** is the location where Experion data is stored. For default installations, this is **C:\ProgramData**.

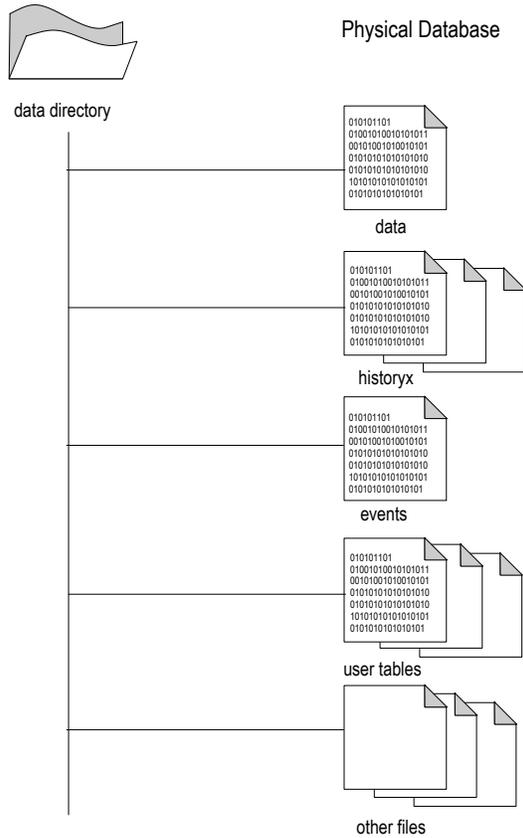
To increase performance, some parts of the database are loaded from the hard disk into the computer's memory when the system starts. Periodically this memory-resident data is written back to the hard disk so that it will not be lost if the system stops.

The database folder contains the following main files.

File	Description
data	Holds many of the smaller database tables and all of the memory-resident tables.
history	Contains the process history data for each history interval.
events	Holds event data.
crtbkr, crtdfd, crtsha	Holds the display definition.
points	Contains all the point and parameter details.

The following figure shows how the database is stored.

## Physical database



## Logical structure

When accessing server data, you will typically work with two types of logical files in the server real-time database:

- *Flat logical files* below

These are arranged as a set of fixed size flat files, containing a fixed number of records, with a fixed number of words per record.

- *Object-based real-time database files* on page 43

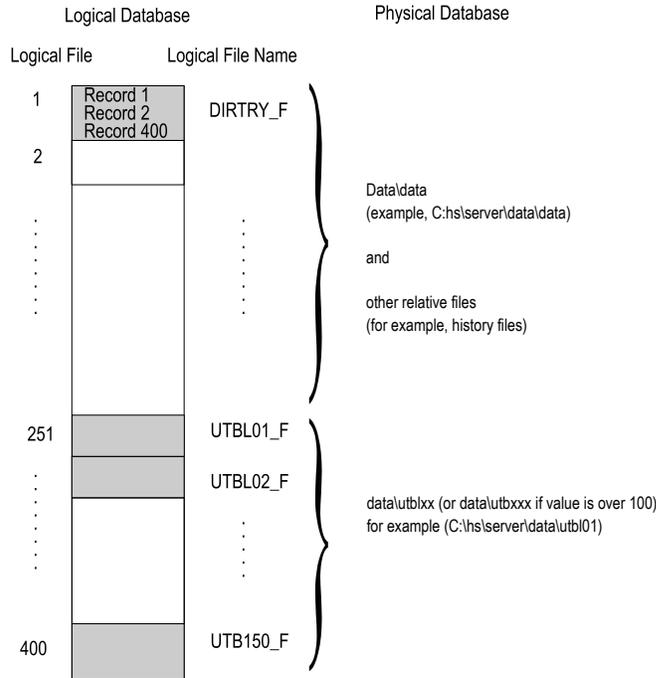
These are flexible data structures for which the underlying structure is hidden from the user and can only be accessed via functions that manipulate the data.

## Flat logical files

To an application, these appear as a set of approximately 400 logical files. Each logical file stores a set of records of data related to some part of the server system.

An example of a logical file is CRTTBL. This table contains a single record for each Station on the system. The records of CRTTBL define items like the type of keyboard connected, the update rate, the current page number and so on.

*Logical versus physical structure of database*



**Flat logical file numbers**

A unique file number (ranging from 1 to 400) is used to identify each of the flat logical files. Labels for all the valid file numbers are defined in the include file **files**. See the C/C++ Application Template section on how to include this file into your application.

Whenever you are reading or writing on a file basis you will need to identify the flat logical file by providing one of these labels as the file number.

**Definition files for flat logical files**

The layout of each of the flat logical files is described in a separate definition file that can be included at the top of your source code. The naming convention for these definitions files is as follows:

- **GBxxxtbl.h** for C/C++ definition files.

Where **xxx** is part of the flat logical file's name.

In our example above the definition file for CRTTBL would be **GBcrttbl.h** for C/C++.

## Types of flat logical files

There are two common types of record structures for the flat logical files used in the server database: *relative* and *circular*. The structure of the logical file determines to some extent how you access the data within the file.

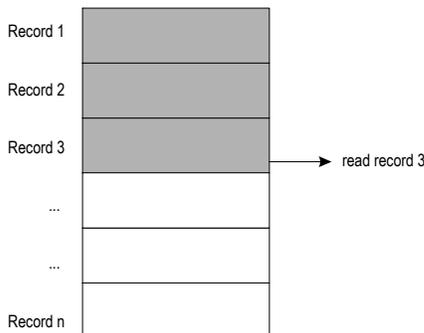
System files have their type determined by Honeywell. User files types are defined when the file is created. To access User files, see 'Accessing user-defined data.'. To create User files, see 'Setting up user tables using the UTBBLD utility.'

### Relative files

Relative files are used where data needs to be stored in a structured way, with each record representing a single, one off entity. An example of a relative file is the CRTTBL where each record represents a single Station. The majority of flat logical files in the server are relative files.

Access to a relative file is achieved using specific functions like **hsc\_param\_value** or using a generic read and write function of DATAIO. If you use DATAIO you will need to provide a record number which is relative to the beginning of the logical file. The record number acts like an index to identify the data—that is, to access data regarding the third Station you would access record three of the logical file CRTTBL.

#### Relative file structure

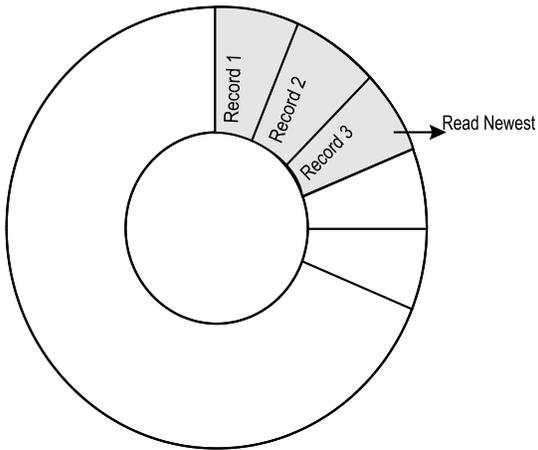


### Circular files

Circular files are used where data needs to be recorded on a regular basis, but there is a limit on the amount of disk space that is to be used. When the circular file is full, and a new record is written to it, the oldest record will be overwritten. An example of a circular file is a HISTORY file where each record represents a set of point parameter values at a given time.

Access to a circular file is achieved using specific functions like **c\_gethstpar\_date\_2** or using the generic functions of DATAIO.

### Circular file structure



### Object-based real-time database files

In addition to the logical files, some data is stored in object-based real-time database files. This data is accessed by various API calls, and the structure of the data is hidden from the user.

An example of such a file is the points database file, whose manipulation functions (or methods) are described in *Accessing acquired data* on the next page.

### Strings in the logical file structure

Only fixed length strings can be stored in the logical file system, because the logical file system uses fixed file record lengths. In addition, the string format in the logical file system is not the same as that in memory representation.

Routines that operate directly against single strings do not need to worry about this detail, as the routines automatically perform the appropriate conversions. However, routines that operate on entire records, such as *c\_dataio\_...()* on page 87, need to be concerned with the format of these strings.

To convert an ANSI string to a fixed length string in a buffer containing a record, you can use the routine *c\_chrint()* on page 85.

To convert to an ANSI string from a fixed length string contained in a buffer representing a record, you can use the routine *c\_intchr()* on page 209.

### Ensuring database consistency

The logical files in the server database are shared by all the tasks and users of the system. This data sharing capability can cause problems if data is not sufficiently protected.

For example, consider the situation where two tasks are simultaneously accessing the same record in a logical file. They both read the record into a buffer in memory and proceed to modify its contents. The first task completes its modification and writes the buffer back to the record in the logical file. A moment later the second task does the same, but it will overwrite the changes made by the first task.

There are two ways to overcome this problem. The first method is to design your tasks so that only one task is responsible for changing the record contents. Because this task is the only one changing records, it can safely read and write as necessary.

The second method is to use file locking. Before performing a read, modify, write sequence your task can call **hsc\_lock\_file** to request permission to change the file. If another task has the file already locked you will be denied access. If the file is not locked, it will be locked on your behalf and you will be able to read, modify and write the record. After you are complete you should call **hsc\_unlock\_file** to allow other tasks to access the file.

Object-based real-time database files do not require such locking. Instead the methods of the file will ensure database consistency.

In most cases you will not need to lock and unlock files or records in your application as the server will perform the necessary locking on your behalf. The exception to this rule is when you are using user tables (see *Accessing user-defined data* on page 50) with more than one task reading and writing to their records. In this case you will need to use the file locking functions of **hsc\_lock\_file** or **hsc\_lock\_record**.

## Accessing acquired data

The data acquired from controllers is stored in an object-based real-time database file, and is accessible to all processes via API calls. The structure of this file is hidden from the API user by the calls used to manipulate it.

### Identifying a point

Before you can access the data from a particular point you need to determine its internal point number. This internal point number is used by several of the application programming library routines to quickly identify the point.

An application will normally determine the internal point number of several points during initialization. To do this the application passes the Point Name in ASCII to the library routine **hsc\_point\_number**. If the point exists in the database, this function will return its corresponding internal point number.

### Identifying a parameter

A point comprises many individual point parameters, for example, SP, OP, PV and so on. When you want to refer to one of these parameters in your application you need to use **hsc\_param\_number** to resolve the parameter name to its appropriate number. This routine accepts an ASCII string for the parameter name and the point number and if the parameter

exists for this point then its corresponding number will be returned. Parameter numbers may vary from point to point (even within the same point type), so parameter names need to be resolved to parameter numbers on a point by point basis.

### Accessing parameter values

To read the current value of a list of parameters, use the **hsc\_param\_values** function, which accepts a list of point and parameter numbers and returns their current value(s).

To write to the value of a particular point's parameter you can call **hsc\_param\_value\_put**, passing it the internal point number, the point's parameter number and the new value. If the parameter has a destination address the Controller will be controlled to the new value. If you do not want such control to be performed use the related write function **hsc\_param\_value\_save**, which performs an identical function but without the control to the parameters destination address.

If the current value for the point is a bad value, then an error code will be returned, and the parameter value you receive will be the last good value for that point's parameter.

### Using point lists

If your application needs to simultaneously read-from or write-to several point parameters, you can create a *point list* which defines the relevant point parameters. (You configure the point list in Station.)

---

#### Attention:

Application point lists only support scan task points.

---

After you have created the point list, your application can use the library routine GETLST to read the set of point parameters, and GIVLST to write the set of point parameters.

For details on the routines, see *c\_getlst()* on page 112 and *c\_givlst()* on page 118 (C and C++).

### To create a point list

1. Choose **ConfigureApplication DevelopmentApplication Point Lists** to call up the Point Lists display.
2. Use the scrollbar to show the point list you want to change.
3. Click the list you want to configure to call up the Application Point List display.
4. For each point parameter you want to control, type the point ID and parameter.
5. If the point parameter is a history parameter you can also define the history offset.

### Controlling when data is acquired and processed for standard points

This section applies only to analog, status, and accumulator points.

Rather than having data acquired on a periodic basis, you can configure specific points to have the data acquired at the request of an application. This is typically done if the acquired data is only used by the application, or if the data is critical in a calculation and it must be the current field value.

If the acquired data need only be used by the application, you can configure the point so that it does not have a PV PERIOD entry. This will mean that no periodic scanning of the value is performed.

There are several library routines that can be used to control when data is acquired and processed from status, analog, and accumulator points:

Routine	Description
SPSW	Scan Point Special and Wait for Completion
SPVW	Scan Point Value and Wait for Completion
SPS	Scan Point Special
SPV	Scan Point Value
PPSW	Process Point Special and Wait for Completion
PPVW	Process Point Value and Wait for Completion
PPS	Process Point Special
PPV	Process Point Value

The routine SPSW will demand a point parameter to be scanned from the field. If the value scanned has changed then the point parameter will be processed (that is, checked for alarm conditions, execute algorithms where necessary, and so on), store the value in the point record, and then return.

The routine SPVW is used when there is no source address for the point parameter. This allows you to point process a calculated value from your application just as if it were scanned from the field, that is, store the value in the PV, and process algorithms, alarms etc.

The routines SPS and SPV are exactly the same as SPSW and SPVW respectively but they return immediately and do not wait for the processing to complete. These are typically used to improve performance if you have several point parameters on the same Controller to demand scan. Call SPS to quickly queue all the point parameters except the last one. Then call SPSW to queue the last point parameter and wait for all to be processed.

The routines PPSW, PPVW, PPS and PPV are again very similar to the previous routines mentioned except that they will always force the processing of the point parameter even if it has not changed. For performance reasons, we do not recommend you to use these routines unless absolutely necessary.

The routine SPV may also effect the performance of the server adversely if used heavily. If you need to perform a lot of point processing of application values you may consider using the user scan task instead.

## Accessing process history

The server can be configured to keep a historical record of acquired data. This historical data can be shown in Station using the standard trend displays or custom charts.

The following kinds of historical data can be retained.

- Standard history stores snapshots of point parameter values at regular intervals. You can choose from the following default standard history collection rates: 1, 2, 5, 10, 30, and 60 minutes. Standard history also calculates average values for the following intervals using the default base collection rate of 1-minute: 6-minutes, 1-hour, 8-hours, and 24-hours.
- Fast history allows the recording of snapshot values at short regular intervals. You can choose from up to 8 different (configurable) intervals, ranging from 1 and 30 seconds).
- Extended history allows the recording of snapshot values at 1-hour, 8-hour, and 24-hour intervals.
- Exception history collects string parameter values at a specified rate but only stores them if the value or quality of that parameter has changed since it was last stored.

Because the API does not support string values, string values from exception history collection will be converted and returned as floating point values. If a value cannot be converted, a bad value is returned.

Note that the intervals for standard, fast, and extended history (but not exception history) can be changed using the **sysbld** utility. For more information about this utility, see the *Server and Client Configuration Guide*.

## Accessing blocks of history

An application can also access the historical data stored in the server database using the library routine **c\_gethstpar\_...\_2**. This routine allows you to retrieve a block of historical values for certain point parameters.

When referencing what history to retrieve you may specify it by either a date and time or by an offset of sample periods from the current time.

## Accessing other data

All other data in the server database (configuration, status, alarm and event data) can be accessed in a more generic fashion.

---

### Attention:

Accessing other data in this way requires knowledge of the internal structures used by the server. Although these are described in the definition files, Honeywell does not guarantee that these formats will not change from release to release of the server.

---

## Accessing logical files

The library routine DATAIO is a generic means of reading and writing to any of the 400 or so logical files in the server database. It allows you to read or write one or more records (in blocks or one at a time) to or from one or more **int2** buffers (one buffer for each record).

You need to refer to the relevant definition file of the record (to determine the internal structure or layout of the record), to access the individual record fields.

When accessing record fields:

- 16 bit integer data can simply be assigned to other variables.
- 32 bit integer data can be equivalenced or accessed via the macros **ldint4()** and **stint4()** in C/C++.
- floating point data can be equivalenced or accessed via the macros **ldreal()** and **streal()** in C/C++.
- double precision floating point data can be equivalenced or accessed via the macros **lddb1()** and **stdb1()** in C/C++.
- strings can be retrieved from the buffer using **INTCHR**.
- strings can be stored into the buffer using **CHRINT**.
- strings can be converted to upper case before storing using **UPPER**.
- dates stored in Julian days can be converted to day, month and year using **GREGOR** and back again using **JULIAN**.

## Accessing memory-resident files

When the logical file is memory-resident you can reference the records by way of global variables rather than using DATAIO routines. These global variables are located in shared memory, and are common to all tasks.

---

### Caution:

Accessing other data using shared memory not only requires knowledge of the internal structures but also requires care. It is very easy to accidentally alter database values just by setting these global variables.

---

Values are read from the memory-resident file simply by assigning the global variable to another variable. Values are written to the memory-resident file simply by setting the value of the global variable. The set value will be written back to disk automatically when the server performs its next checkpoint if the value is stored in a checkpoint file.

In C the variables for the memory-resident files are defined in separate include files called **GBxxxtbl.h**, where *xxx* is the name of the logical file. The global variables are arrays of structures (one structure per record) that have a name of **GBxxxtbl[]**. For example:

```
#include 'src/GBtrbtbl.h'
```

## DIRTRY (The first logical file in the server database)

The first logical file in the server database is a memory-resident file called DIRTRY. It contains one record for every logical file in the server database. Record 1 represents logical file 1, record 2 represents file 2 and so on right up to the last record.

Each of these records defines the attributes of the corresponding logical file. This includes the type of logical file, whether the file is memory-resident, the maximum number of records the file can contain, the number of active records, the record size in words and other data used internally by the server.

For example, if you wanted to determine the number of Stations that have been implemented in your system.

In C/C++ we would reference the global variable **GBdirtry[n-1].actvrc**. The field actvrc contains the number of active records, and the value n represents the nth record of DIRTRY. In this case, n is CRTTBL\_F since we are concerned with the number of Stations defined in the CRTTBL.

```
crtmax = GBdirtry[CRTTBL_F  
- 1].actvrc;
```

## Accessing user-defined data

The server system provides the application developer with 150 database files for application-specific storage. These files are called *user tables* (or *user files*), and are referred to as user tables 1 through to 150, occupying file numbers 251 to 400 respectively.

In order to use these tables with applications, you must first configure the table(s) to be used. This involves specifying the type of table, the number of records in the table, and the size of each record.

The type of table may be either *relative* (direct) or *circular*. The table type is selected during table creation or modification using the table builder utility **UTBBLD**.

Relative tables are linear in structure, and may be indexed via a record number. You can access records 'directly' using the record number.

Circular files are by design accessed in a circular nature, giving you the ability to continually write to a table by incrementing the index, with the actual index looping back to the beginning of the table when the record pointer exceeds the maximum number of records in the table.

The server database has the first three user tables (tables 1-3, database file numbers 251-253) preconfigured to the following values:

Table Number	File Type	Number of Records	Record Size (words)
1	DIRECT	20	128
2	DIRECT	20	128
3	DIRECT	20	128

If you want to configure or modify any of the 150 user tables, you can use the user table builder utility, **UTBBLD**.

## Displaying and modifying user table data

Once they've been setup, Experion gives you the ability to view and modify data within database files using custom displays. Thus if an application uses a user table, you can also create a display to view and/or modify this data.

For details about creating displays, see the *Display Building Guide* (for DSP displays) and the *HMIWeb Display Building Guide* (for HMIWeb Displays).

## Setting up user tables using the UTBBLD utility

The **utbblid** command-line utility can be used to:

- View the existing table configurations
- Configure new user tables

- Modify existing user table configurations
- Delete existing user tables

For usage notes, see *UTBBLD usage notes* on page 57.

---

### UTBBLD example

This example shows a session that uses **utbbld** to carry out its full range of actions, namely:

- Display the existing user table configurations
- Modify the configuration of user table **42**
- Add user table **21**, with the configuration: **CIRCULAR** file type, **64** records, **18** words per record
- Delete user table **4**
- Display the new user table configurations

The session which carried out these actions is as follows:

```
utbbld
System status is OFF-LINE
USER TABLE BUILDER
~~~~ ~~~~~ ~~~~~
Main Menu.
1. Display current user table configuration
2. Modify existing user tables
3. Add user tables
4. Delete user tables
c. Commit changes
q. Quit
Please choose one of the above (default is q):
1

System User Table Configuration
~~~~~ ~~~~~ ~~~~~ ~~~~~
User Table      File Number    File Type      Number of      Words per
Number          Records        Records        Record
1               251            DIRECT         20             128
2               252            DIRECT         20             128
```

---

---

3	253	DIRECT	20	128
4	254	DIRECT	20	12
42	292	CIRCULAR	10	1

Total configured tables = 5. Number of free  
tables = 145

Hit ENTER to continue:

USER TABLE BUILDER

~~~~ ~~~~~~ ~~~~~~

Main Menu.

1. Display current user table configuration
2. Modify existing user tables
3. Add user tables
4. Delete user tables
- c. Commit changes
- q. Quit

Please choose one of the above (default is q):

**2**

Modify One or More User Tables.

The following tables are configured:

1 2 3 4 42

Please enter the user table number you wish to modify,  
or q to return to the main menu (default is q):

**42**

File number selected is 292

The configuration for this user table is:

File type is CIRCULAR

There are 10 records,

And the record size is 1 words.

Do you want the file to be circular?

Please type (y)es, (n)o or ENTER (default is NO)? (Y/N)

**n**

---

---

Direct (relative) File Type

Record Size Is 1 words

Enter required record size

( 1 to 32767 are allowed, or <return> to leave unchanged)

**42** entered.

There are 10 records

Enter required number of records

( 1 to 32767 are allowed, or <return> to leave unchanged)

**42** entered.

The configuration for this user table is:

File type is RELATIVE (DIRECT)

There are 42 records,

And the record size is 42 words.

Are these values OK?

Please type (y)es, (n)o or ENTER (default is YES)

**y**

Do you wish to view/modify another table?

Please type (y)es, (n)o or ENTER (default is NO)

**no**

USER TABLE BUILDER

~~~~ ~~~~~ ~~~~~~

Main Menu.

1. Display current user table configuration

2. Modify existing user tables

3. Add user tables

4. Delete user tables

c. Commit changes

q. Quit

Please choose one of the above (default is q):

**3**

Add a New User Table

---

You may choose the user table number to add, or let the system choose the next available user number for you.

1. Choose the user table number to add
2. Let system choose the new number
- q. Return to the main menu

Please choose one of the above (default is q):

**1**

Currently configured tables are:

1 2 3 4 42

There are 145 free tables remaining, please choose a free user table number (between 1 and 150).

Enter required table number

( 0 to 150 are allowed, or <return> to leave unchanged)

**21** entered.

(This is file number 271).

Do you want the file to be circular?

Please type (y)es, (n)o or ENTER (default is NO)? (Y/N)

**Y**

Circular File Type

Record Size Is 1 words

Enter required record size

( 1 to 32767 are allowed, or <return> to leave unchanged) **18**

entered.

There are 1 records

Enter required number of records

( 1 to 32767 are allowed, or <return> to leave unchanged) **64**

entered.

The configuration for this user table is:

File type is CIRCULAR

---

There are 64 records,  
And the record size is 18 words.

Is this information OK?  
Please type (y)es, (n)o or ENTER (default is YES) (Y/N)

**Y**

Would you like to add more user tables?  
Please type (y)es, (n)o or ENTER (default is NO) (Y/N)

**n**

USER TABLE BUILDER  
~~~~ ~~~~~~ ~~~~~~

Main Menu.

1. Display current user table configuration
2. Modify existing user tables
3. Add user tables
4. Delete user tables
- c. Commit changes
- q. Quit

Please choose one of the above (default is q):

**4**

Delete One or More User Tables.

The following user tables are configured:

1 2 3 4 21 42

Please enter the user table number you wish to  
delete, or q to return to the main menu (default is q):

**4**

File number selected is 254

Do you wish to delete this table?

Please type (y)es, (n)o or ENTER (default is no) (Y/N)

**Y**

WARNING : this will remove all information in this table.

Do you still wish to delete this table ((y)es/(n)o[default])?

---

---

(Y/N)

**Y**

Table has been deleted.

The following user tables are configured:

1 2 3 21 42

Please enter the user table number you wish to delete, or q to return to the main menu (default is q):

**q**

USER TABLE BUILDER

~~~~ ~~~~~ ~~~~~~

Main Menu.

1. Display current user table configuration
2. Modify existing user tables
3. Add user tables
4. Delete user tables
- c. Commit changes
- q. Quit

Please choose one of the above (default is q):

**1**

System User Table Configuration

~~~~~ ~~~~ ~~~~~ ~~~~~~

| User Table Number | File Number | File Type | Number of Records | Words per Record |
|-------------------|-------------|-----------|-------------------|------------------|
| 1                 | 251         | DIRECT    | 20                | 128              |
| 2                 | 252         | DIRECT    | 20                | 128              |
| 3                 | 253         | DIRECT    | 20                | 128              |
| 21                | 271         | CIRCULAR  | 64                | 18               |
| 42                | 292         | DIRECT    | 42                | 42               |

Total configured tables = 5. Number of free tables = 145.

Hit ENTER to continue:

---

---

```
USER TABLE BUILDER
~~~~ ~~~~~ ~~~~~~
Main Menu.
1. Display current user table configuration
2. Modify existing user tables
3. Add user tables
4. Delete user tables
c. Commit changes
q. Quit
Please choose one of the above (default is q):
c

Updating the modified user tables.....
USER TABLE BUILDER
~~~~ ~~~~~ ~~~~~~
Main Menu.
1. Display current user table configuration
2. Modify existing user tables
3. Add user tables
4. Delete user tables
c. Commit changes
q. Quit
Please choose one of the above (default is q):
q
```

---

## UTBBLD usage notes

### Running UTBBLD with the server running/stopped

It is recommended that any changes to user tables using the **utbbld** command be made with the server system stopped. However, the server database service should be left running as **utbbld** requires access to the server database. Making changes to user tables with the server running is not recommended, as doing so can affect applications that are currently accessing the user tables.

### Viewing user table configuration

If you only use **utbbld** for viewing the current configuration, then **utbbld** can be safely executed while the server is running.

To use **utbbld** without stopping the server, use the **-force** option:

```
utbbld -force
```

---

**Attention:**

This option is not recommended because it may disrupt applications that are running, and changes may not be able to be made to files that are in use.

---

## Preservation of existing files

**utbbld** attempts to preserve data in existing user tables. However, any changes to the number of records or the size of the records in the user table might cause loss of data. The user table could become smaller if either the number of records is reduced, or, the number of words per record is reduced.

## Running UTBBLD in a redundant server system

When you make changes to user tables using the **utbbld** command on the primary server, synchronization with the backup server is lost. You need to manually synchronize the servers so that the changes are replicated to the backup server.

After you have synchronized the servers, it is good practice to ensure the user table changes have been correctly replicated to the backup server.

## Using the database scanning software

The database scanning software, **DBSCN**, enables Experion to utilize user table addresses as point source and destination addresses.

In most cases, where a point is required to access the server database, a 'database point' can be built to accomplish this. These database points are best suited to accessing small amounts of data that may be dispersed throughout the server database.

Occasionally, applications require a substantial amount of point data to be derived from the server database. This data would normally reside in user tables. As scanning data using standard database points can result in significant system loading, it should be avoided. Instead, **DBSCN** should be used to provide a more efficient method of scanning point data from the server database.

---

## Working from a Station

The application interface library provides routines that, when working from a Station, enable you to perform the following tasks:

- Generate alarms
- Display messages
- Print files
- Control custom built X-Y charts

### Running a task from a Station

You run a task from a Station using any of the following methods:

- Pressing a Function key
- Selecting a menu item
- Clicking a button on a display
- Answering a prompt
- Calling up a display

Each of these methods of task activation, and the parameters passed in the parameter block, are described in the section titled "Activating a task."

### Routine for generating an alarm

When a task determines some critical condition has occurred, to alert all the operators at each Station you can generate an application alarm or event using the routine `hsc_notif_send()`.

Alarms usually indicate that an abnormal condition has occurred and that some action should be taken by the operator. Alarms can be given one of three priorities; low, high, or urgent. Depending on other alarms in the system and how the alarm is configured, the alarm can appear in the Alarm Zone on each Station and cause the audible annunciator to sound. The alarm can also be printed to an alarm/event printer. All alarms are recorded in the event file.

Events usually indicate that some condition has occurred that needs to be logged or recorded. They are not added to the alarm list but are printed to the alarm printer if it has been configured.

### Routine for using the Station Message and Command Zones

You can use the OPRSTR library routine to display less-important messages on a particular Station's Message Zone.

This routine enables your task to display the following types of messages:

- **Information** messages that remain in the Message Zone until another message appears.
- **Indicator** messages that are automatically cleared after a certain period of time.
- **Prompt** messages that ask the operator to type some information in the Command Zone. When the operator types a response in the Command Zone and presses ENTER, Station activates the task enabling you to retrieve the operator's response by calling OPRSTR in C/C++.

### **Routine for printing to a Station printer**

An application can generate information associated with a particular Station that you want to print to a printer. For this to happen, you need to write the information to an operating system file and use the library routine PRSEND.

PRSEND enables you to queue the operating system file to print on the Demand Report printer associated with the Station. It also enables you to queue the operating system file to print on a specific printer as well.

## Developing user scan tasks

To introduce unsupported controller-like devices into your system, you can either create an OPC server for your device, or you can use the User Scan Task option to write an application which provides an interface between the device and Experion.

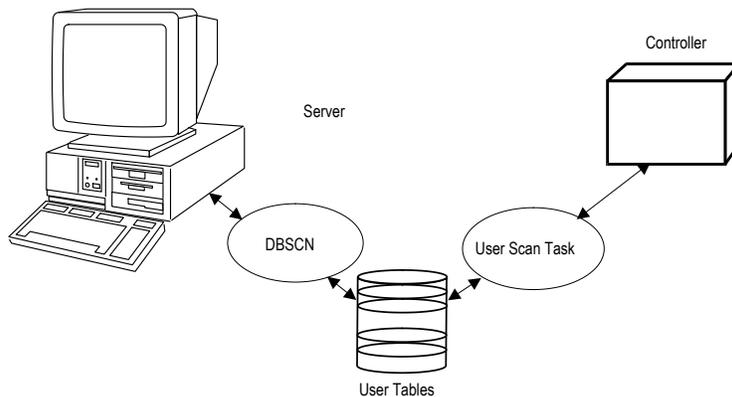
The recommended way is to create an OPC server. This method eliminates the requirement of writing a custom interface by defining a common, high performance interface that permits this work to be done once, and then reused.

You will find the OPC specification on the Internet at: <http://www.opcfoundation.org>. The OPC Specification is a non-proprietary technical specification that defines a set of standard interfaces based upon Microsoft's OLE/COM technology. The application of the OPC standard interface makes possible interoperability between automation/control applications, field systems/devices and business/office applications.

However, should you choose to use the User Scan Task to write an interface application, you will find this option described below.

The link between the server and the User Scan Task is the Experion database user tables. The server provides database scanning software (DBSCN) to scan data from the user tables into server points. The User Scan Task reads data from the remote device and writes it into the user tables. Experion can also send controls to the remote device by way of the User Scan Task.

### *User scan task*



The option works by using channels defined as 'User Scan Task' type channels. These channels operate in exactly the same manner as a conventional channel. They interact, however, with 'User Scan Task' controllers rather than physical controllers.

Point parameters are sourced from these database controllers by specifying the word address within the specific record.

For details about defining a user scan task controller, and its associated channel and points, see the topic 'Creating a user scan task controller' in Quick Builder's help.

## Designing the database for efficient scanning

In order to achieve maximum efficiency when using DBSCN to scan the database, the greatest number of point source addresses should be scanned using the fewest scan packets.

When considering the physical layout of data within a user scan task controller, the following points should be noted:

- Addresses which are scanned at the same rate should be grouped together so that they fall into the same scan packet where possible. The scan processor automatically processes the controller from lowest to highest address and starts a new scan packet each time a scan rate change is detected.
- The number of scan rates in use should be minimized. If only a few points are built for a given scan rate, it may be more efficient to scan them at the next fastest scan rate of many points that are already built on that scan rate.

## Example user scan task

This section includes an example of how to use a User Scan Task.

The example alters the first four values of user table #1 (file number UTBL01\_F) every 10 seconds. It employs many important routines from the application library described in this manual, including DATAIO, GBLOAD, TRM04 and TRMTSK.

The example also demonstrates the use of two important routines, GDBCNT and STCUPD. GDBCNT is used to fetch and decode point control requests from the Station, while STCUPD is used to manipulate the sample time counter used to monitor tasks. For a further description of these routines, see 'Application Library for C and C++.'

Note that the program does not actually communicate with a real physical device. This would be accomplished using standard techniques for accessing the device to be used in the section of the program which is labelled '(**read data from some device or file**)'.

The example provides a C/C++ version of the user-written scan task. It also provides the point and hardware definition files used in conjunction with the program in order to define the database channels, and so on. The example can be found in the folder: **<data folder>\Honeywell\Experion HS\Server\user\examples\src**. Where **<data folder>** is the location where Experion data is stored. For default installations, this is **C:\ProgramData**.

You may want to create a custom display that shows the contents of the first four locations of user table 1 so that table updates can be viewed as they occur. For details, see the *Display Building Guide* (for DSP displays) and the *HMIWeb Display Building Guide* (for HMIWeb displays).

### C/C++ version

The C version of the example User Scan Task is included here. For more information regarding any of the functions called in this program, see 'Application Library for C and C++.'

---

```

/*****
/***** COPYRIGHT © 1996 - 2012 HONEYWELL PACIFIC *****/
/*****/

#include "src/defs.h"
#include "src/environ.h"
#include "src/M4_err.h"
#include "src/dataio.h"
#include "files"
#include "src/trbtbl_def"

#define FILE UTBL03_F /* user file number */
#define RECORD 1 /* user record number */
#define RTU 1 /* user controller number */

#ifndef lint
static char *ident="@(#)c_dbuser.c,v 720.3";
#endif
static char *programe="c_dbuser.c,v";
/*
BEGIN_DOC
-----
C_DBUSER - user scan task for use with DBSCAN
-----
SUMMARY:
Example user scan task
*/

main ()
{

    /*
    DESCRIPTION:

    Add DBUSER as application via application display.
    Give it a user lrn and a user file number.

```

---

DBUSER acquires data and stores the data in a user file.  
DBUSER accepts control requests to write data.  
DBUSER updates controller's Sample Time Counter (watchdog) to keep controller 'healthy'.

-----  
NOTES -  
-----

RETURN VALUES:

FUNCTIONS CALLED:

RELATED FUNCTIONS:

DIAGNOSTICS:

EXAMPLES:

END\_DOC

\*/

```
#define BUFSZ 10000
    int2      buffer[BUFSZ];    /* file buffer */
    struct prm prmblk;          /* task parameter block */
    int2      cntfil;           /* control file number */
    int2      cntrec;           /* control record number */
    int2      cntwrld;          /* control word number */
    int2      cntbit;           /* control bit number */
    int2      cntwid;           /* control data width */
    double    cntval;           /* control value */
    int       tsklrn;           /* task's lrn */
    int       recnum;           /* dataio record number */
    int       errcode;          /* error number */

    // Attach global common
```

---

```

if (c_gbload() == -1)
{
    // GBLOAD FAILED!

    // Retrieve latest error number
    errcode = c_geterrno();

    // Log the result
    c_logmsg(
        progname,
        "214",
        "DBUSER: common load error 0x%x",
        errcode);

    // EXIT the task
    return (errcode);

    // END GBLOAD FAILED!
}

// Find task's LRN
tsklrn = c_getlrn();
if (tsklrn != -1)
{
    // TSKLRN SUCCESS!

    // Start a 10 second timer
    c_tmstrt_cycle ( 10, 1, 0 );

    // Start an infinite loop for the task
    while (TRUE)
    {
        // Check for any control requests first

        // Get database control request
        if (c_gdbcnt(
            &cntfil,

```

---

```

        &cntrec,
        &cntwrd,
        &cntbit,
        &cntwid,
        &cntval
    )== -1)
{
    // GDBCNT FAILED!
    // Only process task requests when
    // there are no control requests

    // Retrieve latest error number
    errcode = c_geterrno();

    // Check and log the error code if other
    // than
    // that the queue is empty
    if (errcode != M4_QEMPTY)
    {
        // Log the result
        c_logmsg(
            progname,
            "236",
            "DBUSER: GDBCNT error 0x%x",
            errcode);
    }

    // Start task request

    // Get parameters from task request block
    if (c_getreq((int2 *)&prmlk)==0)
    {
        // GETREQ SUCCESS!

        // Test the first parameter
        switch (prmlk.param1)
        {

```

---

```
case 1:
    // Start periodic
    requests

    // Perform data
    gathering
    // (read data from
    some device or file)

    // Lock user file
    if (hsc_lock_file
        (FILE,10000) == -1)
    {
        // LOCK FILE FAILED!

        // Retrieve latest
        error number
        errcode =
        c_geterrno();

        // Log the result
        c_logmsg(
        progname,
        "252",
        "DBUSER:
        file %d
        lock error
        0x%x",
        errcode);

        // END LOCK FILE FAILED!
    }
    else
    {
        // LOCK FILE SUCCESS!

        // Read user file
```

---

```
recnum = RECORD;
if (c_dataio_read_
newest(
FILE,
&recnum,
LOC_MEMORY,
buffer,
BUFSZ
)==-1)
{
// DATAIO
READ NEWEST
FAILED!

// Retrieve
latest error
number
errcode =
c_geterrno();

// Log the
result
c_logmsg(
programe,
"262",
"DBUSER:
file %d
record %d
read error
0x%x",
FILE,
recnum,
errcode);
// END DATAIO
READ NEWEST
FAILED!
}
else
```

---

```

{
    //DATAIO
    READ NEWEST
    SUCCESS!

// Update data in user file
// The following is to provide live data
// to dbscan for testing
// (increment and decrement some values
buffer[0] += 10;
buffer[1] -= 10;
buffer[2] += 10;
if (buffer[2]>10000) buffer[2] -= 10000
buffer[3] -= 10;
if (buffer[3]< 0) buffer[3] += 10000

// Write user file
if (c_dataio_write(
    FILE,
    recnum,
    LOC_MEMORY,
    buffer,
    BUFSZ
    )== -1)
{
// DATAIO WRITE FAILED!
// Retrieve latest error number
errcode = c_geterrno();

// Log the result
c_logmsg(
programe,
"283",
"DBUSER: file %d record %d write error
FILE,
recnum,

```

---

```
        errcode);
        // END DATAIO WRITE FAILED!
    }
    else
    {
        // DATAIO WRITE SUCCESS!
        // Update sample time counter
        if (c_stcupd(RTU,65) == -1)
        {
            // STCUPD FAILED!
            // must be >60

            // Retrieve latest error number
            errcode = c_geterrno();

            // Log the result
            c_logmsg(
                progname,
                "290",
                "DBUSER: stcupd error 0x%x",
                errcode);
            // END DATAIO WRITE SUCCESS!
        }
    }
    // END DATAIO READ NEWEST SUCCESS!
}

// Unlock user file
hsc_unlock_file(FILE);

// END LOCK FILE SUCCESS!
}

// END periodic requests
break;

// Continue with switch options
```

---

```

        default:

            // log the result
            c_logmsg(
                progname,
                "301",
                "DBUSER: unknown function %d",
                prmbblk.param1);
            // END switch
        }
        // END GETREQ SUCCESS
    }
else
{
    // GETREQ FAILED!

    // Retrieve latest error number
    errcode = c_geterrno();
    if(errcode != M4_EOF_ERR)
    {
        // Log the result
        c_logmsg(
            progname,
            "308",
            "DBUSER: GETREQ error 0x%x",
            errcode);
    }
    // Terminate the task
    c_trm04(0);
}
}
else
{
    // GDBCNT SUCCESS!
    // Start servicing control requests

    // Log the result

```

---

```

    c_logmsg(
        progname,
        "319",
        "DBUSER: has control request for %d %d %d %d %d %f",
        cntfil,
        cntrec,
        cntwrd,
        cntbit,
        cntwid,
        cntval);

    // Interpret what file/record/word/bit/width means

    // ***Perform required actions here***

    // END GDBCNT SUCCESS!
}
// End While loop
}

//END TSKLRN SUCCESS!
}
else
{
    // TSKLRN FAILED!

    // Log the result
    c_logmsg(
        progname,
        "",
        "Start c_dbuser as a task. Use \"ct\" and supply a user lrn");

    // END TSKLRN FAILED!
}

// Set successful return value

```

---

---

```
return (0);  
  
// END MAIN  
}
```

---

---

## Development utilities

---

**Tip**

Honeywell will supply detailed information and instructions on how to use this command when required.

---

**ADDTSK**

Add application task.

**Synopsis**

```
addtsk namelrn [priority]
```

| Part            | Description                                                              |
|-----------------|--------------------------------------------------------------------------|
| <b>lrn</b>      | The LRN you have chosen for the task, see 'Selecting an LRN for a task.' |
| <b>priority</b> | The priority of task execution (use 0 as a default).                     |
| <b>name</b>     | The executable file name of your task.                                   |

**Description**

This utility loads the executable program identified by name and prepares it for execution. Once loaded the executable becomes a task with the given LRN and priority ready to be activated.

This utility only works with application LRNs, preventing you from accidentally overwriting a server system task. Use *CT* below if you need to use a reserved LRN for your task.

---

**Example**

```
addtsk usrapp 111 0
```

---

**CT**

Create task.

**Synopsis**

```
ct lrnpriority -efn name
```

| Part            | Description                                                              |
|-----------------|--------------------------------------------------------------------------|
| <b>lrn</b>      | The LRN you have chosen for the task, see 'Selecting an LRN for a task.' |
| <b>priority</b> | The priority of task execution (use 0 as a default).                     |
| <b>name</b>     | The executable file name of your task.                                   |

## Description

This utility loads the executable program identified by name and prepares it for execution. Once loaded the executable becomes a task with the given lrn and priority ready to be activated.

Only use this utility if you have run out of application LRNs and you need to use a reserved LRN for your task. It is preferable to use the *ADDTASK* on the previous page utility because it will check that you are not overwriting server system tasks.

### Example

```
ct 111 0 -efn usrapp
```

## databd

### Description

The **databd** command is used to import and export server configuration data in XML format. **databd** is described in detail in the "Server database configuration utility (databd)" section of the *Server and Client Configuration Guide*.

### DBG

Configure Experion so that the next task started from the command line or Visual Studio that calls **gload()** will automatically be assigned the specified LRN.

### Synopsis

```
dbg lrn
```

| Part       | Description                                                              |
|------------|--------------------------------------------------------------------------|
| <b>lrn</b> | The LRN you have chosen for the task, see 'Selecting an LRN for a task.' |

## Description

This utility sets up Experion so that the next manually started task will run with a specified LRN. This is useful for debugging purposes, as it allows a task to be run from within Visual Studio.

---

### Example

```
dbg 111
```

---

## DT

Delete task.

## Synopsis

```
dt lrn
```

| Part       | Description                                                              |
|------------|--------------------------------------------------------------------------|
| <b>lrn</b> | The LRN you have chosen for the task, see 'Selecting an LRN for a task.' |

## Description

This utility marks the specified task for deletion. When the task next calls either TRM04 or TRMTSK, the task will be deleted.

Only use this utility if you have run out of application LRNs and you needed to use a reserved LRN for your task. It is preferable to use the remtsk utility because it will check that you are not removing server system tasks.

---

### Example

```
dt 111
```

---

## ETR

Enter task request

## Synopsis

```
etr lrn [-wait] [-arg arg1]
```

| Part             | Description                                                                                                                                                    |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>lrn</b>       | The LRN you have chosen for the task, see 'Selecting an LRN for a task.'                                                                                       |
| <b>-wait</b>     | Wait for the task to become dormant                                                                                                                            |
| <b>-arg arg1</b> | Additional argument passed to the task via <b>rqstsk</b><br><b>Note:</b> The task is requested via <b>rqstsk</b> —the additional argument can only be an int2. |

## Description

This utility requests the specified task to be activated.

---

### Example

```
etr 111 -arg 5
```

---

## FILDMP

Dump/restore the contents of a logical file.

## Synopsis

```
fildmp
```

## Description

This interactive utility is used to dump, restore or compare the contents of server logical files with standard text files.

When dumping the contents of a logical file to an ASCII operating system file you will need to provide the operating system file name to dump to, the server file number, the range of records to dump, and the data format to dump. Note that the logical file can be dumped to the screen by not specifying an operating system file.

The data format to dump defines how the logical file data will be written to the ASCII operating system file. You can specify INT for integer data, HEX for hexadecimal data, ASC for ASCII data, and FP for floating point data.

When restoring from an operating system file you will only need to provide the operating system file name. The utility will overwrite the current contents of the logical file with what is defined in the operating system file.

### Attention:

Where possible, use the **databd** command instead of **fldmp**. **databd** processes server configuration files in an easier to read format and performs additional validation of the data. However, it is only available for specific server configuration types. For more information, refer to the "Server database configuration utility (databd)" section of the *Server and Client Configuration Guide*.

---

### Example

```
System status is OFF-LINE
Reading from disc. Writing to memory,disc,link.

Enter FUNCTION: 1-dump, 2-restore, 3-compare
1

Enter DEVICE/FILE name
sample.dmp

Enter FILE number
251

Enter START,END record number
1,2

Enter FORMAT: 'INT','HEX','ASC','FP'
HEX

File 251 record 1 dumped
File 251 record 2 dumped
Enter FILE number
Enter FUNCTION: 1-dump, 2-restore, 3-compare
```

---

## FILEIO

Modify contents of a logical file.

### Synopsis

```
fileio
```

### Description

This interactive utility is used to modify the contents of individual fields in a logical file.

You will need to provide the file number, whether to modify memory/disk/both, the record number and the word number of the field to modify and the new value.

---

**Example**

Database contains 400 files  
File number (=0 to exit) ? 251  
Use memory image [YES|NO|BOTH(default)] ?  
File 1 contains 400 records of size 16 words  
Record number (=0 to back up) ? 1  
Word offset (=0 to back up) ? 1  
Mode =0 to back up  
=1 for INTEGER (int2)  
=2 for HEX (int2)  
=3 for ASCII  
=4 for F.P. (real)  
=5 for SET bit  
=6 for CLR bit  
=7 for LONG INTEGER (int4)  
=8 for LONG F.P. (dble) ? 1  
INTEGER VALUE = -32768 NEW VALUE = 100  
Save value [YES|NO(default)] ? YES  
Word offset (=0 to back up) ?  
Record number (=0 to back up) ?  
File number (=0 to exit) ?

---

**REMTSK**

Remove application task.

## Synopsis

remtsk **lrn**

| Part       | Description                                                              |
|------------|--------------------------------------------------------------------------|
| <b>lrn</b> | The LRN you have chosen for the task, see 'Selecting an LRN for a task.' |

## Description

This utility marks the specified task for deletion. When the task next calls either TRM04 or TRMTSK, the task will be deleted.

This utility only works with application LRNs, preventing you from accidentally removing a server system task. Use *DT* on page 76 if you need to use a reserved LRN for your task.

### Example

```
remtsk 111
```

## TAGLOG

List point information

## Synopsis

taglog

## Description

This utility lists information associated with the specified points in the server database. This utility is useful to find out if a point exists and to determine its internal point number.

### Example

An example of output from the utility:

```
Point IPCSTA1      Type 0 Number    1      STALOG PERFORM
TEST 1

DAT file C800 0000 0000 0000 00F0 0000      .....
```

```
EXT file 0000 0000 0000          .....

CNT file 0000 0000 0005 0030 0000 FF00 FFFF 0000 FF00 FFFF
.....0.....
      FF00 FFFF 0000 FF00 FFFF FF00 FFFF FF10 0000 .....

DES file 4950 4353 5441 3120 2020 2020 2020 2020 5354 414C IPCSTA1
      STAL
      4F47 2050 4552 464F 524D 2054 4553 5420 3120 2020 OG PERFORM
TEST 1
      2020 2020 2020 0000 0000 0000 0018 3000 0000 0000
.....0.....
      0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
.....
      0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
.....
      0000 0000 0000 0000 0000 0000
.....
```

---

## USRLRN

Lists LRNs. For details about this utility, see the topic, 'usrlrn', in the .

---

## Application Library for C and C++

The C/C++ application library contains the functions necessary for writing applications that interact with Experion.

### **c\_almmsg\_...()**

Sends a general message for an alarm or event by type.

Note that *hsc\_notif\_send()* on page 146 and *hsc\_insert\_attrib()* on page 131 supersede **c\_almmsg\_...()**.

### **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/almmsg.h>

// Select one of the following synopses as appropriate to
// the type of message being sent

void __stdcall c_almmsg_event(
    char*   text
);

void __stdcall c_almmsg_alarm(
    char*   text,
    int     priority
);

void __stdcall c_almmsg_event_area(
    char*   text,
    char*   area
);

void __stdcall c_almmsg_alarm_area(
    char*   text,
    int     priority,
    char*   area
);

char* __stdcall c_almmsg_format(
    char*   name,
```

```

char*   id,
char*   level,
char*   descr,
char*   value,
char*   units
);
    
```

## Arguments

| Argument        | Description                                                                                                                               |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>text</b>     | (in) pointer to a null-terminated string of text to be sent to the alarm system.                                                          |
| <b>priority</b> | (in) Message priority (see Description).                                                                                                  |
| <b>name</b>     | (in) pointer to a null-terminated string containing the alarm name (40 characters in length (alarm source)).                              |
| <b>id</b>       | (in) pointer to a null-terminated string containing the alarm identifier (for example, PVHI, SVCHG: 20 characters in length (condition)). |
| <b>level</b>    | (in) pointer to a null-terminated string containing the alarm level (for example, U, L, H, STN01).                                        |
| <b>descr</b>    | (in) pointer to a null-terminated string containing the alarm descriptor (132 characters in length).                                      |
| <b>value</b>    | (in) pointer to a null-terminated string containing the alarm value (24 characters in length).                                            |
| <b>units</b>    | (in) pointer to a null-terminated string containing the alarm units (10 characters in length).                                            |
| <b>area</b>     | (in) pointer to a null-terminated string containing the desired of the alarm/event.                                                       |

## Description

Sends the structured text message string to the alarm system for storage into the alarm or event file, and for printing on all printers.

**c\_almmsg\_event** will send the message to all printers and log the text to the event file.

**c\_almmsg\_alarm** will send the message to all printers and log the message to the alarm list or event file. It also sets the first character of the level field of the alarm to either 'L', 'H' or 'U' depending on the value of priority.

The priority of the alarm is defined as follows:

|                   |              |
|-------------------|--------------|
| <b>ALMMSG_LOW</b> | Low priority |
|-------------------|--------------|

|                      |                 |
|----------------------|-----------------|
| <b>ALMMSG_HIGH</b>   | High priority   |
| <b>ALMMSG_URGENT</b> | Urgent priority |

**c\_almmsg\_event\_area** and **c\_almmsg\_alarm\_area** perform the same function as the **c\_almmsg\_event** and **c\_almmsg\_alarm** routines, except that they can be specified.

**c\_almmsg\_format** will format a message given all the relevant fields. It returns a pointer to a null-terminated structured message string that can then be passed onto **c\_almmsg\_event** or **c\_almmsg\_alarm**.

The structured message text string can be broken up into six fields. The starting character of each field is defined by the following identifiers:

|                     |                                           |
|---------------------|-------------------------------------------|
| <b>ALMMSG_NAME</b>  | Alarm name (equals 0)                     |
| <b>ALMMSG_ID</b>    | Alarm ID (for example, PVHI, SVCHG)       |
| <b>ALMMSG_LEVEL</b> | Alarm level (for example, L, U, H, STN01) |
| <b>ALMMSG_DESCR</b> | Alarm description                         |
| <b>ALMMSG_VALUE</b> | Alarm value                               |
| <b>ALMMSG_UNITS</b> | Alarm units                               |

**c\_almmsg\_format2\_malloc** will format a message given all the relevant fields. It returns a pointer to a null-terminated string that can then be passed onto **c\_almmsg\_event** or **c\_almmsg\_alarm**.

For an example of the use of this routine, see *example 2* in **<data folder>\Honeywell\Experion HS\Server\user\examples\src**. Where **<data folder>** is the location where Experion data is stored. For default installations, this is **C:\ProgramData**

## See also

*c\_prsend\_...()* on page 229

## AssignLrn()

Assigns an LRN to the current thread.

## C/C++ Synopsis

```
#include <src/defs.h>
#include <src/trtbl_def>

int2 AssignLrn(
int2* pLrn    // (in/out) lrn to be allocated
```

```

        // -1 == find an unused lrn
        // >0 == allocate the specified lrn
    );

```

## Arguments

| Argument    | Description                                                                                                                                                                                                                                                                                 |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pLrn</b> | <p>(in/out) A pointer to the lrn to be allocated.</p> <p>If *pLrn == -1 then the system will allocate the first available lrn.</p> <p>If *pLrn &gt;0 then the system will use the specified lrn.</p> <p>At the end of a successful call, *pLrn will equal the just assigned lrn number.</p> |

## Description

This function is designed to assign a particular LRN to the current thread. You may choose your own free LRN to use, or you may ask the system to select one for you.

## Diagnostics

This function returns **0** if successful, and **pLrn** will then contain the newly assigned LRN.

## See also

*DeassignLrn()* on page 96

*c\_getlrn()* on page 111

## c\_chrint()

Copies character buffer to integer buffer.

## C/C++ synopsis

```

#include <src/defs.h>
void __stdcall c_chrint(
    char*    chrbuf,
    int      chrbuflen,
    int2*    intbuf,
    int      intbuflen
);

```

## Arguments

| Argument         | Description                                                                             |
|------------------|-----------------------------------------------------------------------------------------|
| <b>chrbuf</b>    | (in) source character buffer containing ASCII                                           |
| <b>chrbuflen</b> | (in) size of character buffer in bytes (to allow non null-terminated character buffers) |
| <b>intbuf</b>    | (out) destination integer buffer                                                        |
| <b>intbuflen</b> | (in) size of destination buffer in bytes                                                |

## Description

Copies characters from a character buffer into an integer buffer. It will either space fill or truncate so as to ensure that **intbuflen** characters are copied into the integer buffer.

If the system stores words with the least significant byte first, then byte swapping will be performed with the copy.

## See also

*c\_intchr()* on page 209

## ctofstr()

Converts a C string to a FORTRAN string.

## C/C++ synopsis

```
#include <src/defs.h>
```

```
void ctofstr(
    char*    Cstr,
    char*    Fstr,
    int      Flen
);
```

## Arguments

| Argument    | Description                                                                    |
|-------------|--------------------------------------------------------------------------------|
| <b>Cstr</b> | (in) null terminated C string                                                  |
| <b>Fstr</b> | (out) memory array for C string ( <b>Fstr</b> can be the same as <b>Cstr</b> ) |
| <b>Flen</b> | (in) length of the Fstr buffer in bytes                                        |

## Description

Given a null terminated C string and the length of the string, this routine will convert it into a FORTRAN string, space padding it if necessary.

If the **Cstr** does not fit, it will be truncated.

## See also

*c\_intchr()* on page 209

*c\_chrint()* on page 85

*ftocstr()* on page 101

## **c\_dataio\_...()**

Routines for accessing the server database logical files.

---

### Tip:

For information about logical files, see 'Logical Structure' and 'Accessing logical files.'

---

The library routine DATAIO is a generic means of reading and writing to any of the 400 or so logical files in the server database. It allows you to read or write one or more records (in blocks or one at a time) to or from an **int2** type buffer.

You need to refer to the relevant definition file of the record (to determine the internal structure or layout of the record), to access the individual record fields.

The following library of DATAIO routines are provided to suit most file record access situations.

When accessing individual records in a flat logical file, the record number of each record remains the same in a relative file (starting with the first record as record 1). However, the record number does not remain the same within a circular file.

If you need to access a particular record in a circular file, you will need to keep track of its physical record number. The READ\_NEWEST and READ\_OLDEST routines are designed to assist in this regard, as they both write the actual record number to a variable provided for the purpose.

The queueing routines are designed to work with circular files, for quickly adding and deleting records to the file. Each use of the QUEUE routine will add a new record unless the file is full, in which case it will overwrite the oldest record with the new record. Each use of the DEQUEUE routine will read and delete the oldest record. In practice, dequeue is not necessary in a circular file because it will automatically overwrite the oldest record when full.

**C/C++ synopsis**

```

#include <src/defs.h>
#include <src/M4_err.h>
#include <src/dataio.h>

// Select one of the following routines as appropriate to
// the type of file and record being accessed

int __stdcall c_dataio_size(
    // retrieves the size of the server file by
    // storing the number of records and
    // storing the number of bytes per record
    int    filenum,    // (in) server file number
    int*   filerecs,  // (out) pointer to the number of records in the file
    int*   byreclen   // (out) pointer to the number of bytes in each record
);

int __stdcall c_dataio_read
(
    // reads a single record into the buffer
    // from a RELATIVE or CIRCULAR server file
    int    filenum,    // (in) server file number
    int    recnum,     // (in) record number
    int    location,   // (in) location of data
    int2*  buffer,     // (in/out) pointer to the buffer
    int    bybuflen   // (in) size of buffer in bytes
);

int __stdcall c_dataio_write(
    // writes a single record from the buffer
    // to a RELATIVE or CIRCULAR server file
    int    filenum,    // (in) server file number
    int    recnum,     // (in) record number
    int    location,   // (in) location of data
    int2*  buffer,     // (in/out) pointer to the buffer
    int    bybuflen   // (in) size of buffer in bytes
);

```

```

int __stdcall c_dataio_read_blk(
    // reads a number of contiguous records
    // from the RELATIVE server file into the buffer
    int    filenum,    // (in) server file number
    int    recnum,    // (in) start record number in block transfer
    int    numrecs,   // (in) number of records in block transfer
    int    location,  // (in) location of data
    int2*  buffer,    // (in/out) pointer to the buffer
    int    bybuflen  // (in) size of buffer in bytes
);

int __stdcall c_dataio_write_blk(
    // writes a number of contiguous records
    // from the buffer to the RELATIVE server file
    int    filenum,    // (in) server file number
    int    recnum,    // (in) start record number in block transfer
    int    numrecs,   // (in) number of records in block transfer
    int    location,  // (in) location of data
    int2*  buffer,    // (in/out) pointer to the buffer
    int    bybuflen  // (in) size of buffer in bytes
);

int __stdcall c_dataio_queue(
    // increments the (internal) newest record pointer
    // and writes a single record from the buffer
    // to the newest record of the CIRCULAR server file
    int    filenum,    // (in) server file number
    int    location,  // (in) location of data
    int2*  buffer,    // (in/out) pointer to the buffer
    int    bybuflen  // (in) size of buffer in bytes
);

int __stdcall c_dataio_dequeue(
    // reads the oldest record into the buffer
    // from the CIRCULAR server file
    // and deletes the oldest record

```

```

    int    filenum,    // (in) server file number
    int    location,  // (in) location of data
    int2*  buffer,    // (in/out) pointer to the buffer
    int    bybuflen   // (in) size of buffer in bytes
);

int __stdcall c_dataio_read_newest(
    // reads the newest record into the buffer
    // from the CIRCULAR server file
    // and stores the actual record number
    int    filenum,    // (in) server file number
    int*   cirrecnum,  // (in/out) pointer to the circular-file record number
    int    location,  // (in) location of data
    int2*  buffer,    // (in/out) pointer to the buffer
    int    bybuflen   // (in) size of buffer in bytes
);

int __stdcall c_dataio_read_oldest(
    // reads the oldest record into the buffer
    // from the CIRCULAR server file
    // and stores the actual record number
    int    filenum,    // (in) server file number
    int*   cirrecnum,  // (in/out) pointer to the circular-file record number
    int    location,  // (in) location of data
    int2*  buffer,    // (in/out) pointer to the buffer
    int    bybuflen   // (in) size of buffer in bytes
);

```

## Arguments

| Argument        | Description                                                                                                                       |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <b>filenum</b>  | (in) Server file number.                                                                                                          |
| <b>filerecs</b> | (out) Pointer to the number of records in the file. The variable referenced in the variable pointer is written to by the routine. |
| <b>byreclen</b> | (out) Pointer to the number of bytes in each record. The variable referenced in the                                               |

| Argument         | Description                                                                                                                                                                                                                                                                                               |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                  | variable pointer is written to by the routine.                                                                                                                                                                                                                                                            |
| <b>cirrecnum</b> | (in/out) Pointer to a variable holding the record number within a circular-file. The variable referenced in the variable pointer must be set prior to calling the routine that uses it to determine which record to access. The variable referenced in the variable pointer is written to by the routine. |
| <b>recnum</b>    | (in) Record number or start record number for block transfer.                                                                                                                                                                                                                                             |
| <b>numrecs</b>   | (in) Number of records to be block transferred.                                                                                                                                                                                                                                                           |
| <b>location</b>  | (in) Location of data                                                                                                                                                                                                                                                                                     |
| <b>buffer</b>    | (in/out) Pointer to the buffer variable. The variable referenced in the variable pointer is written to by the routine.                                                                                                                                                                                    |
| <b>bybuflen</b>  | (in) Size of the buffer in bytes (must be a multiple of 2, because all records are sized in words—2 bytes).                                                                                                                                                                                               |

## Description

Performs data transactions between an application and the server database logical files. Used to read and write server database logical files, records and fields.

---

### Tip:

For an explanation of relative and circular server database logical files, see 'Flat logical files.'

---

|                          |                                                                                                                                                                                                  |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>c_dataio_size</b>     | Retrieves the size of a server file by writing both the number of records and the number of bytes per record to the memory variables referenced by the variable pointers passed-in as arguments. |
| <b>c_dataio_read</b>     | Reads a single record from a server file into the buffer, by writing the record data to the memory variable referenced by the variable pointer passed-in as an argument.                         |
| <b>c_dataio_write</b>    | Writes a single record from the buffer to a server file.                                                                                                                                         |
| <b>c_dataio_read_blk</b> | Reads a number of contiguous records from a server file into the buffer, by writing the record data to the memory variable referenced by the variable pointer passed-in as an argument.          |

|                             |                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>c_dataio_write_blk</b>   | Writes a number of contiguous records from the buffer to a server file.                                                                                                                                                                                                                                                                        |
| <b>c_dataio_queue</b>       | Appends and writes a new single record in the CIRCULAR file by appending to the position above the previous newest record. Writes this new record, and if the file is full, overwrites the oldest record with this new record. Changes the number of records, unless the file is full, in which case it does not change the number of records. |
| <b>c_dataio_dequeue</b>     | Reads and deletes the oldest record in the CIRCULAR file. Changes the number of records. The previously second oldest record then becomes the oldest record. Writes the record data to the memory variable referenced by the variable pointer passed-in as an argument.                                                                        |
| <b>c_dataio_read_newest</b> | Reads any single record in the CIRCULAR file by referencing the records counting from the newest record. Does not change the record data or the number of records. Writes both the record data and the actual record number to the memory variables referenced by the variable pointers passed-in as arguments.                                |
| <b>c_dataio_read_oldest</b> | Reads any single record in the file by referencing the records counting from the oldest record. Does not change the record data or the number of records. Writes both the record data and the actual record number to the memory variables referenced by the variable pointers passed-in as arguments.                                         |

## Location options

The recommended configuration to use is **LOC\_ALL** for most situations, and especially for applications running on a redundant system.

**LOC\_ALL** operates in the most efficient manner possible, first to memory, then to local disk, and finally to the backup server (memory and disk).

**LOC\_MEMORY** and **LOC\_DISK** are only for accessing a file in those specific locations, and that if used, additional file handling must be provided in the custom application to prevent inconsistencies between files in memory, on disk, and on the backup server.

Only ever use **LOC\_MEMORY** or **LOC\_DISK** under specific situations in a custom app where system performance slowdown is occurring, and file access to the local disk or backup server has been identified as the cause of the slowdown.

The location options are listed in the following table:

|                   |                                                                                                                                     |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>LOC_ALL</b>    | Read/write from/to memory, local disk, and backup server. Does not require any additional file handling to ensure file consistency. |
| <b>LOC_MEMORY</b> | Read/write from memory only. Requires additional file handling to ensure file consistency.                                          |
| <b>LOC_</b>       | Read/write from the local disk only. Requires additional file handling to ensure file                                               |

|             |              |
|-------------|--------------|
| <b>DISK</b> | consistency. |
|-------------|--------------|

## Diagnostics

Returns 0 if successful, otherwise, returns -1 if failed and writes an error code to the system error status. Subsequently calling *c\_geterrno()* on page 106 will return one of the following error codes:

|                  |                                      |
|------------------|--------------------------------------|
| [M4_BAD_READ]    | Read error.                          |
| [M4_BAD_WRITE]   | Write error.                         |
| [M4_BAD_FILE]    | Illegal file number.                 |
| [M4_BUF_SMALL]   | Buffer is too small to receive data. |
| [M4_BEYOND_FILE] | Attempt to read outside file.        |
| [M4_RANGE_ERROR] | Size of transfer exceeds 32k.        |
| [M4_ILLEGAL_LFN] | Illegal LFN.                         |
| [M4_NO_BACKUP]   | Backup access not permitted.         |
| [M4_BAD_INTFLG]  | Illegal location value.              |
| [M4_FILE_LOCKED] | File locked to another task.         |

---

### Explanation example

Say, for example, that there was a circular file that contained 10 records numbered 1 to 10 from oldest to youngest with record number 1 being the oldest through to record number 10 being the youngest.

In this example scenario, a call to READ\_OLDEST with a record number argument of "1", would result in the reading of the oldest record, which in this example is actual record number 1. The variable for the record number argument would have had to be holding the value of "1" before the call, and will be holding the value of "1" after the call.

Similarly, a call to READ\_OLDEST with a record number argument of "3", would result in the reading of the third oldest record, which in this example is actual record number 3. The variable for the record number argument would have had to be holding the value of "3" before the call, and will be holding the value of "3" after the call.

---

---

Now compare this with a call to `READ_NEWEST` with a record number argument of "1", which would result in the reading of the newest record, which in this example is actual record number 10. The variable for the record number argument would have had to be holding the value of "1" before the call, and will be holding the value of "10" after the call.

Similarly, a call to `READ_NEWEST` with a record number argument of "3", would result in the reading of the third newest record, which in this example is actual record number 8. The variable for the record number argument would have had to be holding the value of "3" before the call, and will be holding the value of "8" after the call.

---

### Code example

The following example demonstrates how to read and write a record in a User table. It first determines the size of a User table, sizes the buffer appropriately, reads the record, allows for record data manipulation, and finally writes the record back to the User table.

```
#include 'files'           /* for UTBL01's file number */
#include 'applications'    /* for UTBL01's record size */
#include 'src/defs.h'
#include 'src/M4_err.h'
#include 'src/dataio.h'

// ...
// The other code in your application may go here
// ...

// START SERVER FILE ACCESS
// Declare and initialise variables for file record access
int    errcode;
int    rec = 1;
int2   *buffer = 0;
int    user_table1_records = 0;
int    user_table1_records_size = 0;

/* Determine the size of user table UTBL01 */
if (c_dataio_size(    UTBL01_F,
                    &user_table1_records,
                    &user_table1_records_size) == -1)
```

---

---

```
{
// Failed to determine size of user table
// Retrieve latest error number
errcode = c_geterrno();
// Display error message
printf('c_dataio_size error 0x%x', errcode);
// Exit the program and return the error code
exit(errcode);
}
else
{
// Success determining table size
// This data is now stored in the variables:
// 'user_table1_records' and 'user_table1_recordsize'

// Allocate memory for the record buffer
buffer = malloc(user_table1_recordsize);

// Read one record from the disk resident user table UTBL01
if (c_dataio_read(    UTBL01_F,
                    rec,
                    LOC_ALL,
                    buffer,
                    user_table1_recordsize) == -1)
{
    // Failed to read record
    // Retrieve latest error number
    errcode = c_geterrno();
    // Display error message
    printf('c_dataio_read error 0x%x', errcode);
    // Exit the program and return the error code
    exit(errcode);
}
else
{
    // Success reading record into local buffer
    // This data is now stored in the variable 'buffer'
```

---

```
// ...
// Perform data manipulation with 'buffer' here
// ...
// When finished, and if necessary
// Write data back to database
if (c_dataio_write(      UTBL01_F,
                        rec,
                        LOC_ALL,
                        buffer,
                        user_table1_recordsizes) == -1)
{
    // Failed to write record
    // Retrieve latest error number
    errcode = c_geterrno();
    // Display error message
    printf('c_dataio_write error 0x%x', errcode);
}
}
}
// END SERVER FILE ACCESS

// ...
// Do more things in your application here
// ...
```

---

## See also

*hsc\_param\_values()* on page 170

*hsc\_param\_value\_put()* on page 174

*c\_getlst()* on page 112

*c\_givlst()* on page 118

## DeassignLrn()

Removes the current LRN assignment for a thread.

## C/C++ Synopsis

```
#include <src/defs.h>
#include <src/trbtbl_def>
```

```
int2 DeassignLrn();
```

## Description

Removes the association between the thread and its LRN.

## Diagnostics

Upon successful completion, a value of **0** is returned. Otherwise, **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

## See also

*AssignLrn()* on page 84

*c\_getlrn()* on page 111

## c\_deltask()

Marks a task for deletion.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>
```

```
int __stdcall c_deltask(
    int lrn
);
```

## Arguments

| Argument   | Description                                                                               |
|------------|-------------------------------------------------------------------------------------------|
| <b>lrn</b> | (in) Logical resource number of the task to mark for deletion or -1 for the calling task. |

## Description

Marks a task for deletion. After the marked task terminates (by calling *c\_trmsk()* on page 246 or *c\_trm04()* on page 245) it will be deleted from the system.

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned, and calling *c\_geterrno()* on page 106 will return the following error code:

|                         |                                    |
|-------------------------|------------------------------------|
| <b>[M4_ILLEGAL_LRN]</b> | An illegal LRN has been specified. |
|-------------------------|------------------------------------|

---

### Example

```
#include <lrns>      /* for task lrns */
#include <src/M4_err.h>
#include <src/defs.h>
...
int  errcode;
...
/* mark the first user task for deletion on next termination */
if (c_deltask(USR1LRN) == -1)
    {
    errcode = c_geterrno();
    c_logmsg(progname, '123', 'c_deltask error 0x%x', errcode);
    exit(errcode);
    }
```

---

### See also

*c\_trmstk()* on page 246

*c\_trm04()* on page 245

### DbletoPV()

Inserts a double value into a **PARvalue** union.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/almmsg.h>

PARvalue* DbletoPV(
    double          dble_val,
```

```

        PARvalue*      pvvalue
    );

```

## Arguments

| Argument          | Description                                                         |
|-------------------|---------------------------------------------------------------------|
| <b>pvvalue</b>    | (in) A pointer to a <b>PARvalue</b> structure.                      |
| <b>double_val</b> | (in) The double value to insert into the <b>PARvalue</b> structure. |

## Description

Inserts a double value into a **PARvalue**, and then returns a pointer to the **PARvalue** passed in. This function allows you to set attributes into a notification structure using calls to *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138 functions in a single line of code.

## Diagnostics

If this function is successful, the return value is a pointer back to the **PARvalue** passed in, otherwise, the return value is **NULL** and calling *hsc\_insert\_attrib()* on page 131 will retrieve the error code.

Possible errors returned are:

|                         |                                                               |
|-------------------------|---------------------------------------------------------------|
| <b>BUFFER_TOO_SMALL</b> | The pointer to the <b>PARvalue</b> is invalid, that is, null. |
|-------------------------|---------------------------------------------------------------|

## Example

See the examples in *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138.

## See also

*hsc\_insert\_attrib()* on page 131

*hsc\_insert\_attrib\_byindex()* on page 138

*Int2toPV()* on page 206

*Int4toPV()* on page 208

*PritoPV()* on page 228

*RealtoPV()* on page 231

*StrtoPV()* on page 240

*TimetoPV()* on page 242

## **dsply\_lrn()**

Determines the LRN of the display task for a Station, based on the Station number.

### **C/C++ Synopsis**

```
#include <src/defs.h>
```

```
int2 dsply_lrn(  
    int2* pStationNumber // pointer to the Station number  
);
```

### **Arguments**

| <b>Argument</b>       | <b>Description</b>                                                    |
|-----------------------|-----------------------------------------------------------------------|
| <b>pStationNumber</b> | (in) pointer to the Station number that will be used to find the LRN. |

### **Description**

Quickly determines the LRN of a particular Station's display task.

### **Diagnostics**

This function returns the LRN (>0) if successful. Otherwise it returns **-1**.

### **See also**

*stn\_num()* on page 240

## **c\_ex()**

Executes the command line.

### **C/C++ synopsis**

```
#include <src/defs.h>
```

```
int __stdcall c_ex(  
    char*    command  
);
```

## Arguments

| Argument       | Description                                                                      |
|----------------|----------------------------------------------------------------------------------|
| <b>command</b> | (in) pointer to a null-terminated string containing the command line to execute. |

## Description

Passes a command line string as input to the command line interpreter and executes it as if the command line was entered in from a Console Window.

## Diagnostics

If successful completion a value of **0** is returned. Otherwise, **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code relevant to the command line that was executed.

## ftostr()

Converts a FORTRAN string to a C string.

## C/C++ synopsis

```
#include <src/defs.h>
```

```
char* ftoastr(
    char*   from_str,
    int     from_len,
    char*   to_str,
    int     to_len
);
```

## Arguments

| Argument        | Description                                                                          |
|-----------------|--------------------------------------------------------------------------------------|
| <b>from_str</b> | (in) FORTRAN string to convert.                                                      |
| <b>from_len</b> | (in) length of FORTRAN string                                                        |
| <b>to_str</b>   | (out) memory array for C string ( <b>to_str</b> can be the same as <b>from_str</b> ) |
| <b>to_len</b>   | (in) size of array for C string                                                      |

## Description

Given a FORTRAN string and the length of the string, this routine will convert it into a null terminated C string. The string is returned in data buffer supplied.

A pointer to the string is returned if the conversion was successful and NULL pointer is returned if the string passed in (minus trailing blanks) is longer than the output buffer length. In this case a truncated string is returned in the output data buffer.

Danger:

This routine searches from the end of the string for the last non space character. Thus if the string contains something other than spaces on the end of the string, the routine will not work.

If the name to convert is coming from C, then the **strlen** should be passed to this routine rather than the size of the memory allocated for the name.

## See also

*c\_intchr()* on page 209

*c\_chrint()* on page 85

*ctofstr()* on page 86

## c\_gbload()

Loads the global common server database files.

## C/C++ synopsis

```
#include <src/defs.h>

int __stdcall c_gbload();
```

## Description

Makes the server database accessible to the calling task.

The memory-resident sections of the database are attached to the calling task. This allows the calling task to reference the database directly.

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, the application should report an error and terminate. The error code can be subsequently retrieved by calling *c\_geterrno()* on page 106.

## Warnings

Should only be called once per execution of a task, before any other application routines are called.

---

### Example

```
#include <src/defs.h>
#include <src/M4_err.h>

int    errcode;

/* attach to the Server database */
if (c_gbload() == -1)
    {
        errcode = c_geterrno();
        c_logmsg(progname, '123', 'c_gbload error 0x%x', errcode);
        exit(errcode);
    }
```

---

## c\_gdbcnt()

Gets a database control request.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int __stdcall c_gdbcnt(
    int2*    file,
    int2*    record,
    int2*    word,
    int2*    bit,
    int2*    width,
    double*  value
);
```

## Arguments

| Argument      | Description                                                                                          |
|---------------|------------------------------------------------------------------------------------------------------|
| <b>file</b>   | (out) file number that was controlled.                                                               |
| <b>record</b> | (out) record number that was controlled.                                                             |
| <b>word</b>   | (out) word number that was controlled.                                                               |
| <b>bit</b>    | (out) bit number that was controlled.<br>0-15 for int2 data, 0 for int4, float, dble.                |
| <b>width</b>  | (out) width of the data that was controlled. 1-16 for int2 data, 32 for int4 and float, 64 for dble. |
| <b>value</b>  | (out) value to which the file, record, word, bit, width was controlled.                              |

## Description

Used to fetch and decode a control request from the database scan task. See 'Developing user scan tasks.'

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned, and calling **c\_geterrno()** will return one of the following error codes:

|                       |                                                          |
|-----------------------|----------------------------------------------------------|
| [M4_QEMPTY]           | The queue is empty.                                      |
| [M4_ILLEGAL_RTU]      | The Controller number is not legal.                      |
| [M4_ILLEGAL_CHN]      | The channel number is not legal.                         |
| [M4_ILLEGAL_CHN_TYPE] | The channel type is not that of a database scan channel. |

## c\_getapp()

Gets the application record for a task.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/apptbl_def>

int __stdcall c_getapp(
    char*    taskname,
```

```

        uint2    task_lrn,
        struct   apptbl appbuf
    );

```

## Arguments

| Argument         | Description                                                              |
|------------------|--------------------------------------------------------------------------|
| <b>task_name</b> | (in) character string containing the name of the task to find            |
| <b>task_lrn</b>  | (in) logical resource number of the task to find. If -1 then not checked |
| <b>appbuf</b>    | (out) application record buffer as defined in <b>APPTBL_DEF</b>          |

## Description

Finds the corresponding application table record that contains a reference to the specified task. If successful, it will load the record into the supplied **appbuf** and return.

## GetGDAERRcode()

Returns the error code from a **GDAERR** status structure.

## C/C++ synopsis

```

#include <src/defs.h>
#include <src/gdamacro.h>

DWORD GetGDAERRcode (
    GDAERR* pGdaError
);

```

## Arguments

| Argument         | Description                                                        |
|------------------|--------------------------------------------------------------------|
| <b>pGdaError</b> | (in) pointer to the <b>GDAERR</b> structure containing the status. |

## Description

Returns the error code associated with the **GDAERR** structure.

## See also

*IsGDAwarning()* on page 212

*IsGDAerror()* on page 210

*IsGDAnoerror()* on page 211

## **c\_geterrno()**

Returns the error code from an Experion Server API function.

### **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/M4_err.h>
```

```
int c_geterrno();
```

### **Arguments**

None.

### **Returns**

Returns the meaningful error code when aExperion Server API function has been called and has returned **TRUE**. This function needs to be called as the first function immediately after a failed Server API call.

### **Description**

Returns the most recent error code of the Server API function calls. Note that this may not be associated with the most recent Server API function call, but from an earlier call, whichever last returned an error.

You should only ever retrieve the latest Server API error code immediately after testing each Server API function return value for its error status.

---

#### **Attention:**

Any applications that use the function **c\_geterrno()** must include the **M4\_err.h** header file, that is, **#include <src/M4\_err.h>**.

---

#### **Example**

```
#include <src/defs.h>
#include <src/M4_err.h>

int errcode;
```

---

---

```

/* attach to the Server database */
if (c_gblog() == -1)
{
    errcode = c_geterrno();
    c_logmsg(progname, '123', 'c_gblog error 0x%x', errcode);
    exit(errcode);
}

```

---

## **c\_getstpar\_...\_2()**

Gets the history interface parameters.

### **C/C++ synopsis**

```

#include <src/defs.h>
#include <src/M4_err.h>
#include <src/getst.h>

// Select one of the following synopses as appropriate to
// the type of request being sent

int __stdcall c_getstpar_date_2
(
    int     type,
    int     date,
    float   time,
    int     numhst,
    PNTNUM  points,
    PRMNUM  params,
    int     numpnt,
    char*   archive,
    float*  values
);

int __stdcall c_getstpar_ofst_2
(
    int     type,
    int     offset,

```

```

        int      numhst,
        PNTNUM  points,
        PRMNUM  params,
        int      numpnt,
        char*   archive,
        float*  values
    );

```

## Arguments

| Argument       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>    | (in) history type (see Description).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>date</b>    | (in) start date of history to retrieve in Julian days (number of days since 1 Jan 1981).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>time</b>    | (in) start time of history to retrieve in seconds since midnight.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>offset</b>  | (in) offset from latest history value in history intervals (where offset=1 is the most recent history value).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>numhst</b>  | (in) number of history values to be returned per Point.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>points</b>  | (in) array of Point type/numbers to process (maximum of 100 elements).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>params</b>  | (in) array of point parameters to process. Each parameter is associated with the corresponding entry in the points array. The possible parameters are defined in the file 'parameters' in the <b>def</b> folder (maximum 100 elements).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>numpnt</b>  | (in) number of Points to be processed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>archive</b> | <p>(in) pointer to a null-terminated string containing the folder name of the archive files relative to the archive folder. A NULL pointer implies that the system will use current history and any archive files that correspond to the value of the date and time parameters. The archive files are found in <b>&lt;data folder&gt;\Honeywell\Experion HS\Server\data\archive</b>. Where <b>&lt;data folder&gt;</b> is the location where Experion data is stored. For default installations, this is <b>C:\ProgramData</b>.</p> <p>For example, to access the files in <b>&lt;data folder&gt;\Honeywell\Experion HS\Server\archive\ay2012m09d26h11r008</b>, the archive argument is <b>ay2012m09d26h11r008</b>.</p> |
| <b>values</b>  | (out) two dimensional array large enough to accept history values. If there is no history for the requested time or if the data was bad, then -0.0 is stored in the array. Sized <b>numpnt * numhst</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

## Description

Used to retrieve a particular type of history values for specified Points and time in history. History will be retrieved from a specified time or Offset going backwards in time **numhst** intervals for each Point specified.

|                           |                                                                                    |
|---------------------------|------------------------------------------------------------------------------------|
| <b>c_gethstpar_date_2</b> | retrieves history values from a specified date and time.                           |
| <b>c_gethstpar_ofst_2</b> | retrieves history values from a specified number of history intervals in the past. |

The history values are stored in sequence in the **values** array. **values[x][y]** represents the **yth** history value for the **xth** point.

The history type is specified by using one of the following values:

| Value              | Description                       |
|--------------------|-----------------------------------|
| <b>HST_1MIN</b>    | one minute standard history       |
| <b>HST_6MIN</b>    | six minute standard history       |
| <b>HST_1HOUR</b>   | one hour standard history         |
| <b>HST_8HOUR</b>   | eight hour standard history       |
| <b>HST_24HOUR</b>  | twenty four hour standard history |
| <b>HST_5SECF</b>   | Fast history                      |
| <b>HST_1HOURE</b>  | one hour extended history         |
| <b>HST_8HOURE</b>  | eight hour extended history       |
| <b>HST_24HOURE</b> | twenty four hour extended history |

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned, and calling **c\_geterrno()** will return one of the following error codes:

|                         |                                                       |
|-------------------------|-------------------------------------------------------|
| <b>[M4_ILLEGAL_VAL]</b> | Illegal number of Points or history values specified. |
| <b>[M4_ILLEGAL_HST]</b> | Illegal history type or interval specified.           |
| <b>[M4_VAL_NOT_FND]</b> | value not found in history.                           |

---

**Example**

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/gethst.h>
#include "parameters"
#define NHST 50
#define NPNT 3

int errcode; /* error code */
int date; /* julian date */
float time; /* seconds from midnight */
int year; /* year from OAD */
int month; /* month (1 - 12) */
int day; /* day (1 - 31) */
int i; /* iteration */
int hour; /* hour (0 - 23) */
int minute; /* min (0 - 59) */
int type; /* history type */
PNTNUM points[NPNT]; /* point numbers */
PRMNUM params[NPNT]; /* parameters */
float values[NPNT][NHST]; /* history values */
char *programe = "myapp";

. . .
/* set hour, minute, year, month, and day */
year = 2012;
month = 4;
day = 16;
hour = 10;
minute = 0;

. . .
. . .
. . .
/* attach database */
if (c_gbload())
{
    errcode = c_geterrno();
    c_logmsg(programe, "123", "c_gbload error %#x", errcode);
}
```

---

---

```

        exit(errcode);
    }

    /*get the point numbers of the following points*/
    points[0] = hsc_point_number("C1TEMP");
    points[1] = hsc_point_number("C1PRES");
    points[2] = hsc_point_number("C2TIME");

    /*set up for all PV parameters*/
    for (i=0; i<NPNT; i++)
        params[i]=PV;

    /*set up seconds since midnight and julian date*/
    time = (hour* 60+minute)* 60;
    date = c_gtoj(year, month, day);

    /* define history type */
    type = HST_1MIN;
    . . .
    . . .
    . . .
    /*retrieve the history*/
    if (c_gethstpar_date_2(type, date, time, NHST, points, params,
        NPNT, NULL, values[0])
    {
        errcode = c_geterrno();
        c_logmsg(progname,"123"," c_gethstpar_date_2
            error %#x",errcode);
        exit(errcode);
    }
    . . .
    . . .
    . . .

```

---

**See also***hsc\_param\_values()* on page 170**c\_getlrn()**

Gets a logical resource number.

## C/C++ synopsis

```
#include <src/defs.h>

int __stdcall c_getlrn();
```

## Arguments

None.

## Description

Fetches the calling task's Logical Resource Number (LRN). Each LRN is unique for the thread of each process. Each thread can only be associated with one LRN and each LRN can only be associated with one thread.

## Diagnostics

Upon successful completion, the task's LRN is returned. Otherwise, **-1** is returned indicating that the task has not been created as a server task.

## See also

*AssignLrn()* on page 84

*DeassignLrn()* on page 96

## c\_getlst()

Gets values for a list of points.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/lstfil_def>

void __stdcall c_getlst(
    int2    list,
    float*  values,
    int2*   errors
);
```

## Arguments

| Argument         | Description                                                                  |
|------------------|------------------------------------------------------------------------------|
| <b>list</b>      | (in) list number (valid list numbers declared in <b>def/src/lstfil_def</b> ) |
| <b>values[ ]</b> | (out) real array of values of point\parameter list. Sized <b>GGLNM</b> .     |
| <b>errors[ ]</b> | (out) array of returned error codes. Sized <b>GGLNM</b> .                    |

## Description

Used to retrieve values for a list of points and parameters. These point lists can be viewed and modified using the **Application Point Lists** display.

The arrays **values** and **errors** must be large enough to hold the number of items in a list as declared in the parameter **GGLNM** in the file **def/src/listfil\_def**.

## Diagnostics

Upon successful completion zeros will be returned in all elements of the **errors** array. Otherwise one of the following error codes will be returned in the corresponding element of the **errors** array:

|                      |                                                               |
|----------------------|---------------------------------------------------------------|
| [M4_INVALID_NO_ARGS] | An invalid number of parameters was passed to the subroutine. |
| [M4_INV_POINT]       | An invalid point type\number has been specified.              |
| [M4_INV_PARAMETER]   | An invalid parameter has been specified.                      |
| [M4_ILLEGAL_TYPE]    | A parameter with an illegal type has been specified.          |

## See also

*c\_givlst()* on page 118

*c\_dataio\_...()* on page 87

*hsc\_param\_values()* on page 170

*hsc\_param\_value\_put()* on page 174

## **c\_getprm()**

Gets parameters from a queued task request. (Requested via Action Algorithm 92).

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int2 __stdcall c_getprm(
    int    paramc,      //Parameter count (3)
    int2*  par1,        //Parameter 1 value
    int2*  rqstblk,     //Pointer to request block buffer
    int    rqstblk_sz  //Size of request block buffer
);
```

## Arguments

| Argument          | Description                                                         |
|-------------------|---------------------------------------------------------------------|
| <b>paramc</b>     | (in) Parameter count. For standard use this value must be set to 3. |
| <b>par1</b>       | (out) Parameter 1 value.                                            |
| <b>rqstblk</b>    | (out) Pointer to request block buffer.                              |
| <b>rqstblk_sz</b> | (in) Size of request block buffer in bytes.                         |

## Description

Gets parameters from a queued task request. The routine retrieves a parameter block from the request queue. The words in the parameter block are copied to the argument **rqstblk**. If the task is expecting data and the request queue is empty, a value of **M4\_EOF\_ERR** (0x21F) is returned and the task should terminate and wait for the next request. If the parameter block is larger than the size of **rqstblk**, a value of **M4\_RECORD\_LENGTH\_ERR** (0x21A) is returned to indicate the data has been truncated. This routine enables a task to be requested via a point build with Algorithm 92.

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned, and calling *c\_geterrno()* on page 106 will return the following error code: **M4\_ILLEGAL\_LRN** (0x802). The calling process has not been created as an Experion task.

---

### Example

```
#include 'src/defs.h'
#include 'src/M4_err.h'
```

---

```
#define BUFSZ 20
#define FOREVER 1

main()
{
  int2 paramc = 3;
  int2 par1 = 0;
  int2 rqstblk[BUFSZ];
  int2 rqstblk_sz;
  int2 rqst_status;
  int2 status;

  rqstblk_sz = BUFSZ* sizeof(int2);

  while(FOREVER)
  {
    /* get the parameter block for this request */
    rqst_status = c_getprm(&paramc, &par1, (int2 *)rqstblk, &rqstblk_sz);
    if ( rqst_status == M4_EOF_ERR )
    {
      /*terminate and wait for next request*/
      c_trm04(status);
      continue;
    }

    /*****
    /*      Main processing loop      */
    /*                                  */
    /*****/
  }
}
```

---

### Contents of request buffer

The request buffer will be filled with the contents of the requesting points, Algo Block from word 6 of the Algo Block onwards, that is:

rqstblk[0] = Algo Block Word 6 (Task Parameter 1)

rqstblk[1] = Algo Block Word 7 (Task Parameter 2)

In addition the requesting point's point number will be passed in the request buffer:

`rqstblk[3]` = Point number of requesting point.

## See also

`c_rqtskb...()` on page 232

`c_getreq()` below

## c\_getreq()

Gets parameters from task request block.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/trbtbl_def>

int __stdcall c_getreq(
    int2*   prmblk
);
```

## Arguments

| Argument            | Description                      |
|---------------------|----------------------------------|
| <code>prmblk</code> | (out) pointer to parameter block |

## Description

Retrieves a ten word parameter block from the **TRBTBL** of the calling task. If no requests are pending, returns **TRUE (-1)** and calling `c_geterrno()` on page 106 will retrieve the error code **M4\_EOF\_ERR** (0x21F), otherwise, the ten word parameter block is copied into the argument `prmblk`. The parameter block in the **TRBTBL** of the calling task is then cleared and the function returns **FALSE (0)**.

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned, and calling `c_geterrno()` on page 106 will return one of the following error codes:

|                         |                                                            |
|-------------------------|------------------------------------------------------------|
| <b>[M4_ILLEGAL_LRN]</b> | The calling process has not been created as a server task. |
| <b>[M4_EOF_ERROR]</b>   | There are no requests pending.                             |

**Example**

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/trbtbl_def>

main()
{
    int errcode; /* Error number*/
    struct prm prmlk;
    if (c_gblob() == -1)
        errcode = c_geterrno();
    exit(errcode);
    while (1)
    {
        if (c_getreq((int2 *) &prmlk))
        {
            errcode = c_geterrno();
            if (errcode != M4_EOF_ERR)
            {
                /* Report an error */
            }
            /* Now terminate and wait for the */
            /* next request */
            c_trm04(ZERO_STATUS);
        }
        else
        {
            /* Perform some function */
            /* Perhaps switch on the first */
            /* Parameter */
            switch(prmlk.param1)
            {
            case 1:
                ...
                break;
            case 2:
                ...
            }
        }
    }
}
```

---

---

```

                break;
            } /* end switch */
        } /* if */
    } /* end while */
}

```

---

## See also

*c\_rqtskb...()* on page 232

*c\_trm04()* on page 245

## c\_givlst()

Gives values to a list of points.

## C/C++ synopsis

```

#include <src/defs.h>
#include <src/M4_err.h>
#include <src/lstfil_def>

void __stdcall c_givlst(
    int2    list,
    float*  values,
    int2*   errors
);

```

## Arguments

| Argument         | Description                                                           |
|------------------|-----------------------------------------------------------------------|
| <b>list</b>      | (in) list number (valid list numbers declared in <b>lstfil_def</b> )  |
| <b>values[ ]</b> | (in) real array of values of point/parameter list. Sized <b>GGLNM</b> |
| <b>errors[ ]</b> | (out) array of returned error codes. Sized <b>GGLNM</b>               |

## Description

Used to store values into a list of points and parameters and controls those parameters if they have a destination address. Note that each individual parameter control is performed sequentially using a separate scan packet.

These point lists can be viewed and modified using the **Application Point Lists** display.

The arrays **values** and **errors** must be large enough to hold the number of items in a list as declared in the parameter **GGLNM** in the file **lstfil\_def**.

## Diagnostics

Upon successful completion zeros will be returned in all elements of the **errors** array. Otherwise one of the following error codes will be returned in the corresponding element of the **errors** array:

|                             |                                                                               |
|-----------------------------|-------------------------------------------------------------------------------|
| <b>[M4_INVALID_NO_ARGS]</b> | An invalid number of parameters was passed to the subroutine.                 |
| <b>[M4_INV_POINT]</b>       | An invalid point type/number has been specified.                              |
| <b>[M4_INV_PARAMETER]</b>   | An invalid parameter has been specified.                                      |
| <b>[M4_ILLEGAL_TYPE]</b>    | A parameter with an illegal type has been specified.                          |
| <b>[M4_PNT_ON_SCAN]</b>     | It is illegal to store the PV parameter of a point that is currently on scan. |

## See also

*c\_getlst()* on page 112

*c\_dataio\_...()* on page 87

*hsc\_param\_values()* on page 170

*hsc\_param\_value\_put()* on page 174

## **hsc\_asset\_get\_ancestors()**

Gets the asset ancestors for an asset.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_asset_get_ancestors (
    PNTNUM    ,
    int*      piNumAncestors,
    PNTNUM**  ppAncestors
);
```

## Arguments

| Argument       | Description               |
|----------------|---------------------------|
|                | (in) asset point number   |
| piNumAncestors | (out) number of ancestors |
| ppAncestors    | (out) array of ancestors  |

## Description

This functions returns the asset ancestors for the specified asset.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```
#include <src/defs.h>
#include <src/points.h>

int iNumAncestors = 0;
PNTNUM *pAncestors = NULL;

if (hsc_asset_get_ancestors (point, &iNumAncestors, &pAncestors) !=
0)
    return -1
.
.
.
hsc_em_FreePointList (pAncestors);
```

---

## **hsc\_asset\_get\_children()**

Gets the children of an asset.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_asset_get_children(
    PNTNUM    ,
    int*      piNumChildren,
    PNTNUM**  ppChildren
);
```

## Arguments

| Argument      | Description              |
|---------------|--------------------------|
|               | (in) asset point number  |
| piNumChildren | (out) number of children |
| ppChildren    | (out) array of children  |

## Description

Returns the asset children for the specified asset.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```
#include <src/defs.h>
#include <src/points.h>

int iNumChildren = 0;
PNTNUM *pChildren = NULL;

if (hsc_asset_get_children (point, &iNumChildren, &pChildren) != 0)
    return -1
.
```

---

---

```

.
.
hsc_em_FreePointList (pAncestors);

```

---

## **hsc\_asset\_get\_descendents()**

Gets the descendents of an asset.

### **C/C++ synopsis**

```

#include <src/defs.h>
#include <src/points.h>

int hsc_asset_get_descendents (
    PNTNUM    ,
    int*      piNumDescendents,
    PNTNUM**  ppDescendents
);

```

### **Arguments**

| Argument         | Description                 |
|------------------|-----------------------------|
|                  | (in) asset point number     |
| piNumDescendents | (out) number of descendents |
| ppDescendents    | (out) array of descendents  |

### **Description**

Returns the asset descendents for the specified asset.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

### **Diagnostics**

If successful, **0** is returned, otherwise **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

**Example**

```

#include <src/defs.h>
#include <src/points.h>

int iNumDescendents = 0;
PNTNUM *pDescendents = NULL;
if (hsc_asset_get_descendents (point, &iNumDescendents, &pDescendents)
    != 0)
    return -1
.
.
.
hsc_em_FreePointList (pDescendents);

```

---

**hsc\_asset\_get\_parents()**

Gets the parent asset of an asset.

**C/C++ synopsis**

```

#include <src/defs.h>
#include <src/points.h>

int hsc_asset_get_parents(
    PNTNUM    ,
    int*      piNumParents,
    PNTNUM**  ppParents
);

```

**Arguments**

| Argument     | Description             |
|--------------|-------------------------|
|              | (in) asset point number |
| piNumParents | (out) number of parents |
| ppParents    | (out) array of parents  |

## Description

Returns the asset parents for the specified asset.

The array must be cleared by calling *hsc\_em\_FreePointList()* below.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```
#include <src/defs.h>
#include <src/points.h>

int iNumParents = 0;
PNTNUM *pParents = NULL;

if (hsc_asset_get_parents (point, &iNumParents, &pParents) != 0)
    return -1
.
.
.
hsc_em_FreePointList (pParents);
```

---

## **hsc\_em\_FreePointList()**

Frees the memory used to hold a list of points.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_em_FreePointList (
    PNTNUM*   pPointList
);
```

## Arguments

| Argument          | Description                |
|-------------------|----------------------------|
| <b>pPointList</b> | (in) pointer to point list |

## Description

Frees the memory used to hold a list of points.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

## **hsc\_em\_GetLastPointChangeTime()**

Gets the last time a point was changed.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

void hsc_em_GetLastPointChangeTime (
    HSCTIME*    pTime
);
```

## Description

Returns the last time that a point was changed on the server due to a Quick Builder or Enterprise Model Builder download.

## **hsc\_em\_GetRootAlarmGroups()**

Gets the point numbers of the root Alarm Groups.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_em_GetRootAlarmGroups (
    int*    pCount,
```

```

        PNTNUM** ppRootAlarmGroups
    );

```

## Arguments

| Argument          | Description                       |
|-------------------|-----------------------------------|
| pCount            | (out) number of root Alarm Groups |
| ppRootAlarmGroups | (out) array of root Alarm Groups  |

## Description

Returns the point numbers for all of the root Alarm Groups.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```

#include <src/defs.h>
#include <src/points.h>

int iNumRootAlarmGroups = 0;
PNTNUM *pRootAlarmGroups = NULL;
if (hsc_em_GetRootAlarmGroups (&iNumRootAlarmGroups,
    &pRootAlarmGroups) != 0)
    return -1
.
.
.
hsc_em_FreePointList (pRootAlarmGroups);

```

---

## hsc\_em\_GetRootAssets()

Gets the point numbers for root assets.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_em_GetRootAssets (
    int*      pCount,
    PNTNUM** ppRoot
);
```

## Arguments

| Argument | Description          |
|----------|----------------------|
| pCount   | (out) number of root |
| ppRoot   | (out) array of root  |

## Description

Returns the point numbers for all of the root assets.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```
#include <src/defs.h>
#include <src/points.h>

int iNumRoot = 0;
PNTNUM *pRoot = NULL;
if (hsc_em_GetRoot (&iNumRoot, &pRoot) != 0)
    return -1;
.
.
.
hsc_em_FreePointList (pRoot);
```

---

## **hsc\_em\_GetRootEntities()**

Gets the point numbers for all root entities.

### **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/points.h>

int hsc_em_GetRootEntities;(
    int*      pCount,
    PNTNUM** ppRootEntities
);
```

### **Arguments**

| <b>Argument</b> | <b>Description</b>            |
|-----------------|-------------------------------|
| <b>pCount</b>   | (out) number of root entities |
| ppRootEntities  | (out) array of root entities  |

### **Description**

Returns the point numbers for all of the root entities in the enterprise model.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

### **Diagnostics**

If successful, **0** is returned, otherwise **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### **Example**

```
#include <src/defs.h>
#include <src/points.h>

int iNumRootEntities = 0;
PNTNUM *pRootEntities = NULL;
if (hsc_em_GetRootEntities (&iNumRootEntities, &pRootEntities) != 0)
    return -1
.
```

---

---

```

.
.
hsc_em_FreePointList (pRootEntities);

```

---

## **hsc\_enumlist\_destroy()**

Safely destroys an enumlist.

### **C/C++ synopsis**

```

#include <src/defs.h>
#include <src/points.h>

int hsc_enumlist_destroy(
    enumlist** list
);

```

### **Arguments**

| Argument | Description                          |
|----------|--------------------------------------|
| List     | (in) pointer to an enumeration list. |

### **Description**

Deallocates all strings in an enumeration list along with the array itself.

### **Diagnostic**

The return value will be **0** if successful, otherwise **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### **Example**

Retrieve the enumerated list of values for Pntana1's MD parameter and output this list.

```

#include <src/defs.h>
#include <src/points.h>

PNTNUM point;

```

---

---

```

PRMNUM    param;
enumlist* list;
int       i,n;
point = hsc_point_number('Pntana1');
param = hsc_param_number(point,'MD');
n = hsc_param_enum_list_create(point,param, &list);
for(i=0;i<n;i++)
    c_logmsg('example','enum_listcall', '%10s\t%d',list[i].text,list[i].-
value);
    /*process enumlist*/
    hsc_enumlist_destroy (&list);

```

---

## hsc\_GUIDFromString()

Converts a **GUID** from string format to binary format.

### C/C++ synopsis

```

#include <src/defs.h>
#include <src/points.h>

int hsc_GUIDFromString;(
    char*    szGUID,
    GUID*    pGUID
);

```

### Arguments

| Argument | Description                        |
|----------|------------------------------------|
| szGUID   | (in) <b>GUID</b> in string format  |
| pGUID    | (out) <b>GUID</b> in binary format |

### Description

This function converts a **GUID** from string format to binary format.

### Diagnostics

If successful, **0** is returned, otherwise **-1** is returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

## **hsc\_insert\_attrib()**

Sets an attribute value (identified by name) of a notification structure.

Note that this same functionality can be achieved by using index values instead of named attributes through the use of the *hsc\_insert\_attrib\_byindex()* on page 138 function.

### **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/almmsg.h>

int hsc_insert_attrib(
    NOTIF_STRUCT*    notification,
    char*            attribute_name,
    PARvalue*        value,
    int2             type
);
```

### **Arguments**

| <b>Argument</b>       | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>notification</b>   | (in/out) A pointer to the notification structure.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>attribute_name</b> | (in) The name of the attribute to set. See <i>Attribute Names and Index Values</i> on page 135 for a list of attribute names.                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>value</b>          | (in) A pointer to <b>PARvalue</b> that contains the attribute value. <b>PARvalue</b> is a union of data types and its definition is (definition from <b>include/src/points.h</b> ):<br><br><pre>typedef union {     GDAVARIANT    var;     char          text[PARAM_MAX_STRING_LEN+1];     short         int2;     long          int4;     int8          int8;     float         real;     double        dbble;     struct {         long      ord;         char      text[PARAM_MAX_STRING_LEN+1];     } en;     struct {</pre> |

| Argument    | Description                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             | <pre>                 ULONG    cSize; /* size of serialized variant */                 BYTE     *pData; /* pointer to serialized variant */             } server;         HSCTIME time;     } PARvalue;         </pre> |
| <b>type</b> | (in) The value type being passed.                                                                                                                                                                                      |

## Description

Sets an attribute in the notification structure for use with the *hsc\_notif\_send()* on page 146 function to raise or normalize an alarm, event, or message, as appropriate.

The **category** attribute must be the first attribute set and can only be set once within a notification structure. If you do not set **category** as the first attribute, **INV\_CATEGORY** is the return error and the specified attribute is not set. Once you have set the **category** attribute, you can set other attributes.

This function will attempt to convert the attribute value type from the specified type to the default type for that attribute. If this function cannot convert the specified type to the default type, **VALUE\_COULD\_NOT\_BE\_CONVERTED** is the return error. If this function does not know the specified type, **ILLEGAL\_TYPE** is the return error.

This function validates the attribute values for asset and category. If the asset attribute value is invalid, **INV\_AREA** is the return error, and if the category value is invalid, **INV\_CATEGORY** is the return error. This function also validates the attribute values for station and priority. If these attribute values are invalid, **BAD\_VALUE** is the return error.

## Diagnostics

If the function is successful, the return value is **HSC\_OK**, otherwise the return value is **HSC\_ERROR** and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                            |                                                                                                 |
|----------------------------|-------------------------------------------------------------------------------------------------|
| <b>BAD_VALUE</b>           | The specified attribute value is not valid for this attribute.                                  |
| <b>BUFFER_TOO_SMALL</b>    | The pointer to the notification structure buffer is invalid, that is, null.                     |
| <b>INV_ATTRIBUTE</b>       | The specified attribute name does not exist, or you do not have access to manipulate it.        |
| <b>ILLEGAL_TYPE</b>        | The specified type does not exist.                                                              |
| <b>INV_AREA</b>            | The specified asset attribute is not a valid asset.                                             |
| <b>VALUE_COULD_NOT_BE_</b> | The type could not be converted from the specified type to the default type for that attribute. |

|                             |                                                                                                                                                                  |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CONVERTED</b>            |                                                                                                                                                                  |
| <b>ATTR_NOT_IN_CAT</b>      | The specified attribute does not belong to this category. For a list of valid attributes for a category, see <i>Valid Attributes for a Category</i> on page 138. |
| <b>INV_CATEGORY</b>         | The category for this notification has not been set or the passed category value is not a valid category.                                                        |
| <b>CAT_ALREADY_ASSIGNED</b> | The category for this notification has already been set and cannot be reset.                                                                                     |

### Example

The following example creates a notification structure for a system alarm, setting the description to 'Server API Alarm,' the priority to **ALMMMSG\_LOW**, the sub-priority to **0**, and the value to **4**.

```
#include <src/defs.h>
#include "src/almmsg.h"
#include 'src/M4_err.h'

// declare and clear space for notification
NOTIF_STRUCT myNotification;
memset(&myNotification, 0, sizeof(myNotification));

// PARvalue Buffer
PARvalue pvTmp;

// (mandatory) first insert category Attribute (by name)
if (hsc_insert_attrib(&myNotification, "Category", StrtoPV("System
Alarm", &pvTmp), DT_CHAR) == HSC_ERROR)
    c_logmsg ("example",
             "hsc_insert_attrib call",
             "Unable to insert category attribute [%s],
             error code = 0x%x",
             pvTmp.text,
             c_geterrno());

// insert description attribute
if (hsc_insert_attrib(&myNotification, "Description", StrtoPV("Server
API Alarm", &pvTmp), DT_CHAR) == HSC_ERROR)
```

---

```
    c_logmsg ("example",
             "hsc_insert_attrib call",
             "Unable to insert description attribute [%s],
             error code = 0x%x",
             pvTmp.text,
             c_geterrno());

// insert priority of ALMMSG_LOW and
subpriority 0
if (hsc_insert_attrib(&myNotification, "Priority", PritoPV(ALMMSG_LOW,
0, &pvTmp), DT_INT2) == HSC_ERROR)
    c_logmsg ("example",
             "hsc_insert_attrib call",
             "Unable to insert priority attribute [%hd],
             error code = 0x%x",
             pvTmp.int2,
             c_geterrno());

// insert value attribute of 5 and specify type
INT4
if (hsc_insert_attrib(&myNotification, "Value", Int4toPV(5, &pvTmp),
DT_INT4) == HSC_ERROR)
    c_logmsg ("example",
             "hsc_insert_attrib call",
             "Unable to insert value attribute [%d],
             error code = 0x%x",
             pvTmp.int4,
             c_geterrno());
```

---

## See also

*hsc\_insert\_attrib\_byindex()* on page 138

*hsc\_notif\_send()* on page 146

*DbletoPV()* on page 98

*Int2toPV()* on page 206

*Int4toPV()* on page 208

*PritoPV()* on page 228

*RealtovPV()* on page 231

*StrtovPV()* on page 240

*TimetovPV()* on page 242

### Attribute Names and Index Values

The following table lists the attribute names, the index value associated with the attribute name, and the default data type for the attribute.

| Attribute Name                                                                                                                             | Index Value                                                                                                  | Date Type                | Description                                                                                                                                                                                                                                                                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><i>Flexible attribute</i></p> <p>Attribute name, description, and number defined in the <b>SysCfgSum System Attributes</b> display.</p> | <p>ALMEVTFLEXBASEIDX + <i>&lt;Flexible Attribute Number&gt;</i></p> <p>For example, ALMEVTFLEXBASE IDX+5</p> | DT_VAR                   | Flexible values. As the data type is <b>DT_VAR</b> , the optional type argument must be set.                                                                                                                                                                                                                          |
| Action                                                                                                                                     | ALMEVTACTIDX                                                                                                 | DT_CHAR                  | The maximum size is <b>ALMACTUNT_SZ</b> .                                                                                                                                                                                                                                                                             |
| Actor                                                                                                                                      | ALMEVTACTORIDX                                                                                               | DT_CHAR                  | The actor, for example, an operator. The maximum size is <b>ALMEVTACTOR_SZ</b> .                                                                                                                                                                                                                                      |
| Area code                                                                                                                                  | ALMEVTACDIDX                                                                                                 | DT_INT2<br>or<br>DT_CHAR | <p>If you specify <b>DT_CHAR</b>, you must specify the asset name. If you specify <b>DT_INT2</b>, you must specify the asset number.</p> <p>Must be a valid asset. If no area code attribute is created within the notification, the <b>hsc_notif_send</b> function assigns the system asset to the notification.</p> |
| Asset                                                                                                                                      | ALMEVTASTIDX                                                                                                 | DT_CHAR                  |                                                                                                                                                                                                                                                                                                                       |
| Category                                                                                                                                   | ALMEVTCATIDX                                                                                                 | DT_INT4<br>or<br>DT_CHAR | If you specify <b>DT_CHAR</b> , you must specify the category name. If you specify <b>DT_INT4</b> , you must specify the category index.                                                                                                                                                                              |

| Attribute Name  | Index Value         | Date Type | Description                                                                                      |
|-----------------|---------------------|-----------|--------------------------------------------------------------------------------------------------|
| Changed Time    | ALMEVTCHANGETIMEIDX | DT_TIME   |                                                                                                  |
| Comment         | ALMEVTCOMMENTIDX    | DT_CHAR   | The maximum size is <b>ALMEVTCOMMENT_SZ</b> .                                                    |
| Condition       | ALMEVTCONIDX        | DT_CHAR   | The maximum size is <b>ALMEVTCON_SZ</b> .                                                        |
| Connection      | ALMEVTCONNECTIDX    | DT_INT2   |                                                                                                  |
| Description     | ALMEVTDESIDX        | DT_CHAR   | A description. The maximum size is <b>ALMEVTDES_SZ</b> .                                         |
| Field Time      | ALMEVTETIMEIDX      | DT_TIME   |                                                                                                  |
| Field Time Bias | ALMEVTFIELDBIASIDX  |           |                                                                                                  |
| Flags           | ALMEVTFLAGSIDX      | DT_INT2   |                                                                                                  |
| Limit           | ALMEVTLIMIDX        | DT_DBLE   | The alarm limit.                                                                                 |
| Link 1          | ALMEVTLINK1IDX      | DT_CHAR   | A navigation link. The maximum size is <b>ALMEVTLINK_SZ</b> .                                    |
| Link 1 Type     | ALMEVTLINK1TYPEIDX  | DT_INT2   | Set to the default value when Link 1 is set. Link types are defined in the <b>almmsg.h</b> file. |
| Link 2          | ALMEVTLINK2IDX      | DT_CHAR   | A navigation link. The maximum size is <b>ALMEVTLINK_SZ</b> .                                    |
| Link 2 Type     | ALMEVTLINK2TYPEIDX  | DT_INT2   | Set to the default value when Link 2 is set. Link types are defined in the <b>almmsg.h</b> file. |
| Link 3          | ALMEVTLINK3IDX      | DT_CHAR   | A navigation link. The maximum size is <b>ALMEVTLINK_SZ</b> .                                    |
| Link 3 Type     | ALMEVTLINK3TYPEIDX  | DT_INT2   | Set to the default value when Link 3 is set. Link types are defined in the <b>almmsg.h</b> file. |

| Attribute Name      | Index Value          | Date Type                | Description                                                                                                                                                      |
|---------------------|----------------------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Location Full Name  | ALMEVTLOCFULLNAMEIDX | DT_CHAR                  |                                                                                                                                                                  |
| Location Tag Name   | ALMEVTLOCTAGNAMEIDX  | DT_CHAR                  |                                                                                                                                                                  |
| Prev Value          | ALMEVTPREVVALIDX     | DT_VAR                   |                                                                                                                                                                  |
| Prev Value Type     | ALMEVTPREVVALTYPEIDX | DT_INT2                  |                                                                                                                                                                  |
| Priority            | ALMEVTPRIIDX         | DT_INT2                  | Includes both the priority and sub-priority value. Use the <b>PritoPV</b> function to set this attribute. Both the priority and sub-priority values must be set. |
| Quality             | ALMEVTQUALIDX        | DT_INT2                  | OPC Quality value. Default value set to <b>c0</b> if not set.                                                                                                    |
| Severity            | ALMEVTSEVIDX         | DT_INT4                  | The OPC severity.                                                                                                                                                |
| Signature Meaning 1 | ALMEVTSIGNMEANIDX    | DT_CHAR                  | The maximum size is <b>ALMEVTSIGMEAN_SZ</b> .<br><i>Pharma license only.</i>                                                                                     |
| Source              | ALMEVTSRCIDX         | DT_CHAR                  | The point name.<br>The maximum size is <b>ALMEVTSRC_SZ</b> .                                                                                                     |
| Station             | ALMEVTSTNIDX         | DT_INT2<br>or<br>DT_CHAR | If you specify <b>DT_INT2</b> , the string will be formatted. Otherwise, <b>DT_CHAR</b> is assumed. Must be a valid station.                                     |
| Subcondition        | ALMEVTSUBCONIDX      | DT_CHAR                  | The maximum size is <b>ALMEVTCON_SZ</b> .                                                                                                                        |
| Time                | ALMEVTTIMEIDX        | DT_TIME                  |                                                                                                                                                                  |
| Time Bias           | ALMEVTTIMEBIASIDX    |                          |                                                                                                                                                                  |
| Units               | ALMEVTUNTIDX         | DT_CHAR                  | The maximum size is <b>ALMEVTUNT_SZ</b> .                                                                                                                        |

| Attribute Name | Index Value      | Date Type | Description |
|----------------|------------------|-----------|-------------|
| Value          | ALMEVTVALIDX     | DT_VAR    |             |
| Value Type     | ALMEVTVALTYPEIDX |           |             |

### Valid Attributes for a Category

The following table shows default association of attributes available in categories. Only attribute names indicated with an X can be set for each category.

You can view the categories, and the attributes available in that category, in the `syscfgsumsystemcategories` system display.

### `hsc_insert_attrib_byindex()`

Sets an attribute (identified by its index value) of a notification structure.

Note that this same functionality can be achieved by using named attributes instead of index values through the use of the `hsc_insert_attrib()` on page 131 function.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/almmsg.h>

int hsc_insert_attrib_byindex(
    NOTIF_STRUCT*    notification,
    int2              attribute_index,
    PARvalue*        value,
    int2              type
);
```

### Arguments

| Argument               | Description                                                                                                                                                                               |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>notification</b>    | (in/out) A pointer to the notification structure.                                                                                                                                         |
| <b>attribute_index</b> | (in) The index value of the attribute to insert. See <i>Attribute Names and Index Values</i> on page 135 for a list of index values.                                                      |
| <b>value</b>           | (in) A pointer to <b>PARvalue</b> that contains the attribute value. <b>PARvalue</b> is a union of data types and its definition is (definition from <code>include/src/points.h</code> ): |

| Argument    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             | <pre> typedef union {     GDAVARIANT    var;     char          text[PARAM_MAX_STRING_LEN+1];     short         int2;     long          int4;     int8         int8;     float        real;     double       dble;     struct {         long      ord;         char      text[PARAM_MAX_STRING_LEN+1];     } en;     struct {         ULONG     cSize; /* size of serialized variant */         BYTE      *pData; /* pointer to serialized variant */     } server;     HSCTIME      time; } PARvalue; </pre> |
| <b>type</b> | (in) The value type being passed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## Description

Sets an attribute in the notification structure for use with the *hsc\_notif\_send()* on page 146 function to raise or normalize an alarm, event, or message, as appropriate.

The **ALMEVTCATIDX** (category) attribute must be the first attribute set and can only be set once within a notification structure. If you do not set **ALMEVTCATIDX** as the first attribute, **INV\_CATEGORY** is the return error and the specified attribute is not set. Once you have set the **ALMEVTCATIDX** attribute, you can set other attributes.

This function will attempt to convert the attribute value type from the specified type to the default type for that attribute. If this function cannot convert the specified type to the default type, **VALUE\_COULD\_NOT\_BE\_CONVERTED** is the return error. If this function does not know the specified type, **ILLEGAL\_TYPE** is the return error.

This function validates the attribute values for asset and category. If the asset attribute value is invalid, **INV\_AREA** is the return error, and if the category value is invalid, **INV\_CATEGORY** is the return error. This function also validates the attribute values for station and priority. If these attribute values are invalid, **BAD\_VALUE** is the return error.

## Diagnostics

If the function is successful, the return value is **HSC\_OK**, otherwise the return value is **HSC\_ERROR** and calling `c_geterrno()` on page 106 will retrieve the error code.

The possible errors returned are:

|                                     |                                                                                                                                                                  |
|-------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>BAD_VALUE</b>                    | The specified attribute value is not valid for this attribute.                                                                                                   |
| <b>BUFFER_TOO_SMALL</b>             | The pointer to the notification structure buffer is invalid, that is, null.                                                                                      |
| <b>INV_ATTRIBUTE</b>                | The specified attribute name does not exist, or you do not have access to manipulate it.                                                                         |
| <b>ILLEGAL_TYPE</b>                 | The specified type does not exist.                                                                                                                               |
| <b>INV_AREA</b>                     | The specified asset attribute is not a valid asset.                                                                                                              |
| <b>VALUE_COULD_NOT_BE_CONVERTED</b> | The type could not be converted from the specified type to the default type for that attribute.                                                                  |
| <b>ATTR_NOT_IN_CAT</b>              | The specified attribute does not belong to this category. For a list of valid attributes for a category, see <i>Valid Attributes for a Category</i> on page 138. |
| <b>INV_CATEGORY</b>                 | The category for this notification has not been set or the passed category value is not a valid category.                                                        |
| <b>CAT_ALREADY_ASSIGNED</b>         | The category for this notification has already been set and cannot be reset.                                                                                     |

---

### Example

The following example creates a notification structure for a system alarm, setting the description to 'Server API Alarm,' the priority to **ALMMSG\_LOW**, the sub-priority to **0**, and the value to **4**.

```
#include <src/defs.h>
#include "src/almmsg.h"

// declare and clear space for notification
NOTIF_STRUCT myNotification;
memset(&myNotification, 0, sizeof(myNotification));
```

---

---

```
// PARvalue Buffer
PARvalue pvTmp;

// (mandatory) first insert category Attribute (by name)
if (hsc_insert_attrib_byindex (&myNotification,
ALMEVTCATIDX, StrtoPV("System Alarm", &pvTmp), DT_CHAR) == HSC_ERROR)
    c_logmsg ("example",
              "hsc_insert_attrib call",
              "Unable to insert category attribute [%s],
              error code = 0x%x",
              pvTmp.text,
              c_geterrno());

// insert description attribute
if (hsc_insert_attrib_byindex(&myNotification,
ALMEVTDESIDX, StrtoPV("Server API Alarm", &pvTmp), DT_CHAR) == HSC_
ERROR)
    c_logmsg ("example",
              "hsc_insert_attrib call",
              "Unable to insert description attribute [%s],
              error code = 0x%x",
              pvTmp.text,
              c_geterrno());

// insert priority of ALMMSG_LOW and subpriority 0
if (hsc_insert_attrib_byindex(&myNotification,
ALMEVTPRIIDX, PritoPV(ALMMSG_LOW, 0, &pvTmp), DT_INT2) == HSC_ERROR)
    c_logmsg ("example",
              "hsc_insert_attrib call",
              "Unable to insert priority attribute [%hd],
              error code = 0x%x",
              pvTmp.int2,
              c_geterrno());

// insert value attribute of 5 and specify type
INT4if (hsc_insert_attrib_byindex(&myNotification,
ALMEVTVALIDX, Int4toPV(5, &pvTmp), DT_INT4) == HSC_ERROR)
```

---

---

```

c_logmsg ("example",
          "hsc_insert_attrib call",
          "Unable to insert value attribute [%d],
          error code = 0x%x",
          pvTmp.int4,
          c_geterrno());

```

---

**See also***DbletoPV()* on page 98*hsc\_insert\_attrib()* on page 131*hsc\_notif\_send()* on page 146*Int2toPV()* on page 206*Int4toPV()* on page 208*PritoPV()* on page 228*RealtoPV()* on page 231*StrtoPV()* on page 240*TimetoPV()* on page 242**hsc\_IsError()**

Determines whether a returned status value is an error.

**C/C++ synopsis**

```

#include <src/defs.h>
#include <src/M4_err_def>

```

```

int hsc_IsError(
    int code
);

```

**Arguments**

| Argument    | Description                    |
|-------------|--------------------------------|
| <b>code</b> | (in) The status code to check. |

## Description

Determines whether a particular status code is an error. Most C functions indicate success by returning a 0 in the return value. If a function returns a non-zero value, calling `c_geterrno()` on page 106 will retrieve the error code. This value can then be checked to see if it indicates an error or warning.

Status values can indicate an error, a warning, or success. Some functions return a **GDAERR** structure instead. Use the macro `IsGDAerror()` on page 210 to check this value for an error.

## Diagnostics

This routine returns **TRUE (-1)** if **code** indicates an error condition, otherwise it returns **FALSE (0)**.

## See also

`hsc_IsWarning()` below

`IsGDAerror()` on page 210

## hsc\_IsWarning()

Determines whether a returned status value is a warning.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int hsc_IsWarning(
    int    code
);
```

## Arguments

| Argument          | Description                    |
|-------------------|--------------------------------|
| <code>code</code> | (in) The status code to check. |

## Description

Determines whether a particular status code is warning. Most C functions indicate success by returning a 0 in the return value. If a function returns a non-zero value, calling `c_geterrno()` on page 106 will retrieve the error code. This value can then be checked to see if it indicates an error or warning.

Status values can indicate an error, a warning, or success. Some functions return a **GDAERR** structure instead. Use the macro *IsGDAerror()* on page 210 to check this value for an error.

## Diagnostics

This routine returns **TRUE (-1) code** indicates a warning condition, otherwise it returns **FALSE (0)**.

## See also

*hsc\_IsError()* on page 142

*IsGDWarning()* on page 212

## hsc\_lock\_file()

Locks a database file.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int hsc_lock_file(
    int file,
    int delay
);
```

## Arguments

| Argument     | Description                                                    |
|--------------|----------------------------------------------------------------|
| <b>file</b>  | (in) server file number.                                       |
| <b>delay</b> | (in) delay time in milliseconds before lock attempt will fail. |

## Description

Performs advisory locking of database files. Advisory locking means the tasks which use the file must take responsibility for setting and removing locks as required.

For more information regarding database locking see *Ensuring database consistency* on page 43.

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                 |                                         |
|-----------------|-----------------------------------------|
| <b>[FILLCK]</b> | File locked to another task             |
| <b>[RECLCK]</b> | Record locked to another task           |
| <b>[DIRLCK]</b> | File's directory locked to another task |
| <b>[BADFIL]</b> | Illegal file number specified           |

## See also

*hsc\_lock\_record()* below

*hsc\_unlock\_file()* on page 202

*hsc\_unlock\_record()* on page 203

## hsc\_lock\_record()

Locks a record of a database file.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>
```

```
int hsc_lock_record(
    int file,
    int record,
    int delay
);
```

## Arguments

| Argument      | Description                                                    |
|---------------|----------------------------------------------------------------|
| <b>file</b>   | (in) server file number.                                       |
| <b>record</b> | (in) record number (see description).                          |
| <b>delay</b>  | (in) delay time in milliseconds before lock attempt will fail. |

## Description

Performs advisory locking of database record. Advisory locking means the tasks which use the record must take responsibility for setting and removing locks as required.

For more information regarding database locking see 'Ensuring database consistency.'

## Diagnostics

Upon successful completion a value of 0 is returned. Otherwise, -1 is returned and calling `c_geterrno()` on page 106 will retrieve one of the following error codes:

|           |                                 |
|-----------|---------------------------------|
| [FILLCK]  | File locked to another task     |
| [RECLCK]  | Record locked to another task   |
| [DIRLCK]  | Folder locked to another task   |
| [BADFIL]  | Illegal file number specified   |
| [BADRECD] | Illegal record number specified |

## See also

`hsc_lock_file()` on page 144

`hsc_unlock_file()` on page 202

`hsc_unlock_record()` on page 203

## `hsc_notif_send()`

Sends a notification structure to raise or normalize an alarm, event, or message.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/almmsg.h>

int hsc_notif_send(
    NOTIF_STRUCT*    notification,
    NOTIF_SEND_MODE mode
);
```

## Arguments

| Argument            | Description                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>notification</b> | (in) A pointer to the notification structure.                                                                                                                                                                                                                                                                                                                                                                          |
| <b>mode</b>         | (in) The mode to send the notification. <ul style="list-style-type: none"> <li>■ <b>RAISE</b> sends the notification in the unacknowledged and off-normal state.</li> <li>■ <b>RAISE_NORMALIZED</b> sends the notification in the unacknowledged and normal state.</li> <li>■ <b>NORMALIZE</b> changes the state of a previous notification with identical source and condition, from off-normal to normal.</li> </ul> |

## Description

Sends a notification (as created using the *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138 functions), for storing in the alarm file, or event file, or message directory, as appropriate to the category set in the notification structure.

Note that if a Station printer has been configured to print alarms, events, or messages, this notification will also be printed as appropriate.

This function validates the attribute values of the notification structure for asset, tagname, and Station. If not explicitly set, this function sets default values.

## Diagnostics

If the function is successful, the return value is **HSC\_OK**, otherwise the return value is **HSC\_ERROR** and calling *c\_geterrno()* on page 106 will retrieve one of the following errors:

|                         |                                                                                            |
|-------------------------|--------------------------------------------------------------------------------------------|
| <b>BUFFER_TOO_SMALL</b> | The pointer to the notification is invalid, that is, null.                                 |
| <b>INV_CATEGORY</b>     | The notification does not have a valid category set.                                       |
| <b>M4_QEMPTY</b>        | The file could not be queued to the printer because the printer queue has no free records. |

## Example

This example sends an unacknowledged and off-normal alarm and then returns that alarm to normal.

---

```
#include <src/defs.h>
#include <src/almmsg.h>

// declare and clear space for notification
NOTIF_STRUCT myNotification;
memset(&myNotification, 0, sizeof(myNotification));

// PARvalue Buffer
PARvalue pvTmp;

// (mandatory) first insert category Attribute (by name)
if (hsc_insert_attrib(&myNotification, "Category", StrtoPV("System
Alarm", &pvTmp), DT_CHAR) == HSC_ERROR)
    c_logmsg ("example",
              "hsc_insert_attrib call",
              "Unable to insert category attribute [%s],
error code = 0x%x",
              pvTmp.text,
              c_geterrno());

// insert description attribute
if (hsc_insert_attrib(&myNotification, "Description", StrtoPV("Server
API Alarm", &pvTmp), DT_CHAR) == HSC_ERROR)
    c_logmsg ("example",
              "hsc_insert_attrib call",
              "Unable to insert description attribute [%s],
error code = 0x%x",
              pvTmp.text,
              c_geterrno());

// insert priority of ALMMMSG_HIGH and sub-priority 0
if (hsc_insert_attrib(&myNotification, "Priority", PritoPV(ALMMMSG_
HIGH, 0, &pvTmp), DT_INT2) == HSC_ERROR)
    c_logmsg ("example",
              "hsc_insert_attrib call",
              "Unable to insert priority attribute [%hd],
error code = 0x%x",
```

---

---

```
        pvTmp.int2,
        c_geterrno());

// insert value attribute of 5 and specify type INT4
if (hsc_insert_attrib(&myNotification, "Value", Int4toPV(5, &pvTmp),
DT_INT4) == HSC_ERROR)
    c_logmsg ("example",
        "hsc_insert_attrib call",
        "Unable to insert value attribute [%d],
error code = 0x%x",
        pvTmp.int4,
        c_geterrno());

// insert source of "API call"
if (hsc_insert_attrib(&myNotification, "Source", StrtoPV("API Call",
&pvTmp), DT_CHAR) == HSC_ERROR)
    c_logmsg ("example",
        "hsc_insert_attrib call",
        "Unable to insert source attribute [%s],
error code = 0x%x",
        pvTmp.text,
        c_geterrno());

// insert condition of "APICALL"
if (hsc_insert_attrib(&myNotification, "Condition", StrtoPV("APICALL",
&pvTmp), DT_CHAR) == HSC_ERROR)
    c_logmsg ("example",
        "hsc_insert_attrib call",
        "Unable to insert source attribute [%s],
error code = 0x%x",
        pvTmp.text,
        c_geterrno());

// send notification in unacked and off-normal state
if (hsc_notif_send(&myNotification, RAISE) == HSC_ERROR)
    c_logmsg ("example",
        "hsc_notif_send call",
```

---

---

```

        "hsc_notif_send failed with error code = 0x%x",
        c_geterrno());

// return this alarm to normal
if (hsc_notif_send(&myNotification, NORMALIZE) == HSC_ERROR)
    c_logmsg ("example",
             "hsc_notif_send call",
             "hsc_notif_send failed with error code = 0x%x",
             c_geterrno());

```

---

**See also***hsc\_insert\_attrib()* on page 131*hsc\_insert\_attrib\_byindex()* on page 138*DbletoPV()* on page 98*Int2toPV()* on page 206*Int4toPV()* on page 208*PritoPV()* on page 228*RealtoPV()* on page 231*StrtoPV()* on page 240*TimetoPV()* on page 242**hsc\_param\_enum\_list\_create()**

Get an enumerated list of parameter values.

**C/C++ synopsis**

```

#include <src/defs.h>
#include <src/points.h>

int hsc_param_enum_list_create
(
    PNTNUM    point,
    PRMNUM    param,
    enumlist** list
);

```

## Arguments

| Argument     | Description                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>point</b> | (in) point number                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>param</b> | (in) point parameter number                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>list</b>  | <p>(in/out) pointer to an enumeration list of parameters <b>enumlist</b> is defined as (definition from <b>include/src/dictionary.h</b>):</p> <pre>typedef struct {     int    value;     char*  text; } enumlist;</pre> <ul style="list-style-type: none"> <li>■ <b>value</b> is the ordinal value of the enumeration</li> <li>■ <b>text</b> is the null terminated string containing the enumeration text</li> </ul> |

## Description

Returns a list of enumeration strings for the point parameter value, where applicable.

## Diagnostics

The return value will be the number of entries in the list, otherwise **-1** if an error was encountered and calling **c\_geterrno()** will retrieve the error code.

In all cases the enumlist structure is created by *hsc\_param\_enum\_list\_create()* on the previous page with enough space for the text field in each enumlist element in the enumlist array. Because this memory is allocated by the function, your user code needs to free this space when you finish using the structure. As these functions always allocate the memory required for the text field, make sure that you free all memory before calling the routines a second time with the same enumlist\*\* pointer, otherwise there will be a memory leak. To facilitate freeing this memory, *hsc\_enumlist\_destroy()* on page 129 has been added to the API.

---

### Example

Retrieve the enumerated list of values for Pntana1's MD parameter and output this list.

---

---

```
#include <src/defs.h>
#include <src/points.h>

PNTNUM    point;
PRMNUM    param;
enumlist* list;
int       i,n;

point = hsc_point_number('Pntana1');
param = hsc_param_number(point,'MD');
n = hsc_param_enum_list_create(point,param, &list);
for(i=0;i<n;i++)
    c_logmsg('example',
            'enum_listcall',
            '%10s\t%d',
            list[i].text,
            list[i].value);
/*process enumlist*/
hsc_enumlist_destroy (&list);
```

---

## See also

*hsc\_param\_enum\_ordinal()* below

*hsc\_enumlist\_destroy()* on page 129

## **hsc\_param\_enum\_ordinal()**

Get an enumeration's ordinal value.

## **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/points.h>

int hsc_param_enum_ordinal(
    PNTNUM    point,
    PRMNUM    param,
    char*     string
);
```

## Arguments

| Argument      | Description                 |
|---------------|-----------------------------|
| <b>point</b>  | (in) point number           |
| <b>param</b>  | (in) point parameter number |
| <b>string</b> | (in) enumeration string     |

## Description

Returns the ordinal value that corresponds to the enumeration string for the point parameter.

## Diagnostics

Returns the ordinal number on success and **-1** if an error was encountered. The error code can be retrieved by calling *c\_geterrno()* on page 106.

---

### Example

Determine the ordinal number of the enumeration 'AUTO' for 'MD' parameter for point 'Pntana1.'

```
#include <src/defs.h>
#include <src/points.h>
#include <src/M4_err.h>

PNTNUM    point;
PRMNUM    param;
int4      ordinal;

point = hsc_point_number('Pntana1');
param = hsc_param_number(point, 'MD');
if((ordinal=hsc_param_enum_ordinal (point,param,'AUTO'))<0)
    c_logmsg('example', 'ord call',
            'call to hsc_param_enum_ordinal failed,
            error code = %d',
            c_geterrno());
else
    c_logmsg('example','ord call',
            'Ordinal value for AUTO is %d.',ordinal);
```

---

## **hsc\_param\_enum\_string()**

Gets an enumeration string.

### **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/points.h>

char* hsc_param_enum_string(
    PNTNUM    point,
    PRMNUM    param,
    int4      ordinal
);
```

### **Arguments**

| <b>Argument</b> | <b>Description</b>             |
|-----------------|--------------------------------|
| <b>point</b>    | (in) point number              |
| <b>param</b>    | (in) point parameter number    |
| <b>ordinal</b>  | (in) enumeration ordinal value |

### **Description**

Returns the enumeration string that corresponds to the ordinal value for the point parameter.

### **Diagnostic**

Returns the enumeration string, or **NULL** and calling *c\_geterrno()* on page 106 will retrieve the error code.

The enumeration string must be freed by the caller using the system call **free()**.

## **hsc\_param\_format()**

Gets a parameter's format.

### **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/points.h>

int hsc_param_format (
```

```

        PNTNUM    point,
        PRMNUM    param
    );

```

## Arguments

| Argument     | Description           |
|--------------|-----------------------|
| <b>point</b> | (in) point number     |
| <b>param</b> | (in) parameter number |

## Description

This routine will return the format of the specified point parameter, and will be one of the following, or negative if invalid:

```

DF_CHAR,      /* character          */
DF_NUM,       /* numeric            */
DF_POINT,     /* point name         */
DF_PARAM,     /* parameter name     */
DF_ENG,       /* engineering units  */
DF_PCT,       /* percent            */
DF_ENUM,      /* enumerated          */
DF_MODE,      /* enumerated mode     */
DF_BIT,       /* TRUE/FALSE         */
DF_STATE,     /* state descriptor   */
DF_PNTTYPE,   /* point type         */
DF_TIME,      /* time                */
DF_DATE,      /* date                */
DF_DATE_TIME, /* time stamp         */
DF_GETVAL     /* format as pnt-param */

```

---

### Example

Determines the data format of the parameter **PointDetailDisplayDefault** of point 'pntana1' and outputs this format's value.

```

#include <src/defs.h>
#include <src/points.h>

PRMNUM param;

```

---

---

```
PNTNUM point;
int paramFormat;

point = hsc_point_number("pntana1");
param = hsc_param_number(point, "PointDetailDisplayDefault");
if((paramFormat = hsc_param_format(point, param)) < 0)
    c_logmsg ("example","param_format call",
             "Error getting param format for point %d,
             param %d",
             point,
             param);
else
    c_logmsg ("example",
             "param_format call",
             "Param format of point %d,
             parameter %d is %d",
             point,
             param,
             paramFormat);
```

---

## See also

*hsc\_param\_type()* on page 165

## **hsc\_param\_limits()**

Get parameter data entry limits.

## **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/points.h>

int hsc_param_limits (
    PNTNUM    point,
    PRMNUM    param,
    double*   min,
    double*   max
);
```

## Arguments

| Argument     | Description           |
|--------------|-----------------------|
| <b>point</b> | (in) point number     |
| <b>param</b> | (in) parameter number |
| <b>min</b>   | (out) minimum value   |
| <b>max</b>   | (out) maximum value   |

## Description

Returns the minimum and maximum data entry limits of the specified point parameter.

## Diagnostics

This function always returns **0**. If an error occurs, **min** will be set to **0.0** and **max** to **100.0**.

### Example

Find the parameter limits for point 'pntana1' and parameter 'SP' and output them.

```
#include <src/defs.h>
#include <src/points.h>

PRMNUM param;
PNTNUM point;
double limitMin, limitMax;

point = hsc_point_number("pntana1");
param = hsc_param_number(point, "SP");
if(hsc_param_limits(point, param, &limitMin, &limitMax) != 0)
    c_logmsg ("example",
              "param_limits call",
              "Error getting param limits for point %d,
              param %d",
              point,
              param);
else
    c_logmsg ("example",
```

---

```

    "param_limits call",
    "Param limits of point %d,
    parameter %d are %f -> %f ",
    point,
    param,
    limitMin,
    limitMax);

```

---

## See also

*hsc\_param\_type()* on page 165

## hsc\_param\_subscribe()

Subscribe to a list of point parameters.

## C/C++ synopsis

```

#include <src/defs.h>
#include <src/points.h>

```

```

int hsc_param_subscribe(
    int      number,
    PNTNUM*  points,
    PRMNUM*  param,
    int      period
);

```

## Arguments

| Argument      | Description                      |
|---------------|----------------------------------|
| <b>number</b> | (in) number of entries in lists  |
| <b>points</b> | (in) list of point numbers       |
| <b>params</b> | (in) list of parameter numbers   |
| <b>period</b> | (in) subscription period (msecs) |

## Description

Declares interest in point parameters so that data will be available in the point record, without the need to fetch it from the appropriate location.

## Diagnostics

This function will return **0** if successful, otherwise the relevant status code will be returned.

## See also

*hsc\_param\_values()* on page 170

## hsc\_param\_list\_create()

Gets a list of parameters.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_param_list_create(
    PNTNUM      point,
    enumlist**  list
);
```

## Arguments

| Argument     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>point</b> | (in) point number, specify 0 for all parameters of all point types                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>list</b>  | <p>(in/out) pointer to an enumeration list of parameters <b>enumlist</b> is defined as (definition from <b>include/src/dictionary.h</b>):</p> <pre>typedef struct {     int    value;     char*  text; } enumlist;</pre> <ul style="list-style-type: none"> <li>■ <b>value</b> is the parameter number if the parameter is currently stored in the server database. A zero value may indicate a parameter has not previously been accessed. To obtain the parameter number use <b>hsc_param_number</b>.</li> <li>■ <b>text</b> is the null terminated string containing the parameter name</li> </ul> |

## Description

Returns pointer to a list of names and numbers for the point's parameters.

## Diagnostics

The return value of this function indicates the number of parameters stored in the list structure.

In all cases the enumlist structure is created by *hsc\_param\_list\_create()* on the previous page with enough space for the text field in each enumlist element in the enumlist array. Because this memory is allocated by the function, your user code needs to free this space when you finish using the structure. As these functions always allocate the memory required for the text field, make sure that you free all memory before calling the routines a second time with the same enumlist\*\* pointer, otherwise there will be a memory leak. To facilitate freeing this memory, *hsc\_enumlist\_destroy()* on page 129 is included in the API.

## Example

Retrieves all the parameters for point 'pntana1', and print out the name.

```
#include <src/defs.h>
#include <src/points.h>
#define LISTSZ 1000

enumlist* list;
int n,i;
PNTNUM point;

point = hsc_point_number('pntana1');
n = hsc_param_list_create(point, &list);
for (i=0; i<n; i++)
    c_logmsg ('example',
              'param_list call',
              'parameter _s is %5d',
              list[i].text,
              list[i].value);
```

## See also

*hsc\_enumlist\_destroy()* on page 129

*hsc\_param\_number()* on page 162

## **hsc\_param\_name()**

Get a parameter name.

### **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/points.h>

int hsc_param_name(
    PNTNUM    point,
    PRMNUM    param,
    char*      name,
    int        namelen
);
```

### **Arguments**

| <b>Argument</b> | <b>Description</b>         |
|-----------------|----------------------------|
| <b>point</b>    | (in) point number          |
| <b>param</b>    | (in) parameter number      |
| <b>name</b>     | (out) parameter name       |
| <b>namelen</b>  | (in) length of name string |

### **Description**

The parameter name is returned for the parameter number of the point specified. Note that the char buffer needs to be big enough to store the parameter name in it, and that this length (of the buffer) must be passed in the function.

---

### **Example**

The parameter name for the parameter numbered 16 is returned for point 'pntana1', and prints it out.

```
#include <src/defs.h>
#include <src/points.h>

PNTNUM point;
PRMNUM param;
```

---

---

```

char paramName[MAX_PARAM_NAME_LEN+1];

point = hsc_point_number("pntana1");
param = 16;
hsc_param_name(point,param,paramName,MAX_PARAM_NAME_LEN+1);
c_logmsg ("example",
          "param_list call",
          "Parameter %s is parameter number %d for point %s.",
          paramName,
          param,
          point);

```

---

**See also***hsc\_param\_number()* below**hsc\_param\_number()**

Gets the parameter's number.

**C/C++ synopsis**

```

#include <src/defs.h>
#include <src/points.h>

```

```

PRMNUM hsc_param_number (
          PNTNUM    point,
          char*     name
);

```

**Arguments**

| Argument     | Description         |
|--------------|---------------------|
| <b>point</b> | (in) point number   |
| <b>name</b>  | (in) parameter name |

**Description**

Returns the number of the named point parameter. If the point number is zero, then ALL point types will be searched.

## Diagnostics

If the parameter cannot be found or an error occurs **0** will be returned, and calling *c\_geterrno()* on page 106 will retrieve the error code.

If the point number is zero then all points are searched for the corresponding parameter name. This will then return the first match to any fixed parameters of points in the system. It will not, however, resolve a flexible parameter name to a number, as this parameter number is specific to a point not all points.

### Example

The parameter number for the parameter **PointDetailDisplayDefault** is returned for point 'pntana1,' and is output.

```
#include <src/defs.h>
#include <src/points.h>

PNTNUM point;
PRMNUM param;
char *paramName = "PointDetailDisplayDefault";

point = hsc_point_number("pntana1");
param = hsc_param_number(point, paramName);
c_logmsg ("example",
          "param_number call",
          "Parameter %s is parameter number %d for point number %d.",
          paramName,
          param,
          point);
```

### See also

*hsc\_param\_name()* on page 161

### **hsc\_param\_range()**

Get parameter data range.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_param_range(
    PNTNUM    point,
    PRMNUM    param,
    double*   min,
    double*   max
);
```

## Arguments

| Argument     | Description           |
|--------------|-----------------------|
| <b>point</b> | (in) point number     |
| <b>param</b> | (in) parameter number |
| <b>min</b>   | (out) minimum value   |
| <b>max</b>   | (out) maximum value   |

## Description

Returns the minimum and maximum range of the specified point parameter.

## Diagnostics

This function always returns **0**. If an error occurs, **min** will be set to **0.0** and **max** to **100.0**.

---

### Example

Find the parameter ranges for point 'pntana1' and parameter 'SP' and output them.

```
#include <src/defs.h>
#include <src/points.h>

PRMNUM param;
PNTNUM point;
double rangeMin, rangeMax;
```

---

```
point = hsc_point_number("pntana1");
param = hsc_param_number(point, "SP");
if(hsc_param_range(point, param, &rangeMin, &rangeMax) != 0)
    c_logmsg ("example",
              "param_range call",
              "Error getting param range for point %d,
              param %d",
              point,
              param);
else
    c_logmsg ("example",
              "param_range call",
              "Param range of point %d,
              parameter %d is %f -> %f",
              point,
              param,
              rangeMin,
              rangeMax);
```

---

## See also

*hsc\_param\_limits()* on page 156

## **hsc\_param\_type()**

Get a parameter's data type.

## **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/points.h>

int hsc_param_type(
    PNTNUM    point,
    PRMNUM    param
);
```

## Arguments

| Argument     | Description           |
|--------------|-----------------------|
| <b>point</b> | (in) point number     |
| <b>param</b> | (in) parameter number |

## Description

This routine will return the data type of the specified point parameter, and will be one of the following, or negative if invalid:

```
DT_CHAR      /* character string                */
DT_INT2      /* 1 to 16 bit short integer                 */
DT_INT4      /* 1 to 32 bit long integer                  */
DT_REAL      /* short float                               */
DT_DBLE      /* long float                                */
DT_HIST      /* history (-0 => large float)               */
DT_VAR       /* variant                                   */
DT_ENUM      /* enumeration '                             */
DT_DATE_TIME /* timestamp (integer*2 day, double sec)     */
DT_TIME      /* date and time (HSCTIME format)           */
DT_INT8      /* 64-bit integer                           */
DT_SRCADDR   /* source address                            */
DT_DSTADDR   /* destination address                       */
```

---

### Example

Determine the data type of the parameter **PointDetailDisplayDefault** of point 'pntana1' and output this type's value.

```
#include <src/defs.h>
#include <src/points.h>

PRMNUM param;
PNTNUM point;
int paramType;

point = hsc_point_number("pntana1");
param = hsc_param_number(point, "PointDetailDisplayDefault");
if((paramType = hsc_param_type(point, param)) < 0)
```

---

---

```

        c_logmsg ("example",
                "param_type call",
                "Error getting param type for point %d,
                param %d",
                point,
                param);
    else
        c_logmsg ("example",
                "param_type call",
                "Parameter type of point %d,
                parameter %d is %d",
                point,
                param,
                paramType);

```

---

**See also**

*hsc\_param\_format()* on page 154

**hsc\_param\_value()**

Get a point parameter value.

**C/C++ synopsis**

```

#include <src/defs.h>
#include <src/points.h>

```

```

int hsc_param_value(
    PNTNUM    point,
    PRMNUM    param,
    int*      offset,
    PARvalue* value,
    uint2*    type
);

```

**Arguments**

| Argument     | Description       |
|--------------|-------------------|
| <b>point</b> | (in) point number |

| Argument      | Description                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>param</b>  | (in) parameter number                                                                                                                                                                                                                                                                                                                                                                       |
| <b>offset</b> | (in) point parameter offset (for history parameters).                                                                                                                                                                                                                                                                                                                                       |
| <b>value</b>  | <p>(out) value union</p> <p>PARvalue is a union of data types and is defined as follows (definition from include/src/points.h):</p> <pre>typedef union {     short    int2;     long     int4;     float    real;     double   dble;     char     text[PARAM_MAX_STRING_LEN+1];     struct   {         long   ord;         char   text[PARAM_MAX_STRING_LEN+1];     } en; } PARvalue;</pre> |
| <b>type</b>   | (out) value data type (defined in the <b>parameters</b> file)                                                                                                                                                                                                                                                                                                                               |

## Description

The parameter's definition is located and used to access the point record using a common routine which returns the pointer to the data. The top level routine then extracts the value by type.

It is recommended that *hsc\_param\_values()* on page 170 be used in preference to this function, as it allows the subscription period to be specified.

If your system uses *dynamic scanning*, *hsc\_param\_value()* calls from the Server API do not trigger dynamic scanning.

## Diagnostics

**0** will be returned if successful, otherwise **-1** will be returned, and calling **c\_geterrno()** will retrieve the error code. The value returned in **type** will be one of the following constants defined in the **parameter** file:

```
DT_CHAR      /* character string          */
DT_INT2      /* 1 to 16 bit short integer            */
DT_INT4      /* 1 to 32 bit long integer             */
```

```

DT_REAL      /* short float */
DT_DBL      /* long float */
DT_HIST      /* history (-0 => large float) */
DT_VAR       /* variant */
DT_ENUM      /* enumeration ' */
DT_DATE_TIME /* timestamp (integer*2 day, double sec) */
DT_TIME      /* date and time (HSCTIME format) */
DT_INT8      /* 64-bit integer */
DT_SRCADDR   /* source address */
DT_DSTADDR   /* destination address */
    
```

### See also

*hsc\_param\_values()* on the next page

*hsc\_param\_number()* on page 162

*hsc\_point\_number()* on page 198

### hsc\_param\_value\_of\_type()

Get a point parameter value of specified type.

### C/C++ synopsis

```

#include <src/defs.h>
#include <src/points.h>

int hsc_param_value_of_type(
    PNTNUM      point,
    PRMNUM      param,
    int*        offset,
    PARvalue*   value,
    uint2*      type
);
    
```

### Arguments

| Argument      | Description                                           |
|---------------|-------------------------------------------------------|
| <b>point</b>  | (in) point number                                     |
| <b>param</b>  | (in) parameter number                                 |
| <b>offset</b> | (in) point parameter offset (for history parameters). |

| Argument     | Description                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>value</b> | <p>(out) value union</p> <p>PARvalue is a union of data types and is defined as follows (definition from include/src/points.h):</p> <pre>typedef union {     short    int2;     long     int4;     float    real;     double   dble;     char     text[PARAM_MAX_STRING_LEN+1];     struct   {         long   ord;         char   text[PARAM_MAX_STRING_LEN+1];     } en; } PARvalue;</pre> |
| <b>type</b>  | (out) value data type (defined in the <b>parameters</b> file)                                                                                                                                                                                                                                                                                                                               |

## Description

The parameter's definition is located and used to access the point record using a common routine which returns the pointer to the data. The top level routine then extracts the value by type.

If your system uses *dynamic scanning*, `hsc_param_value_of_type()` calls from the Server API do not trigger dynamic scanning.

## See also

`hsc_param_values()` below

`hsc_param_number()` on page 162

`hsc_point_number()` on page 198

## hsc\_param\_values()

Get multiple point parameter values.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>
```

```

int hsc_param_values(
    int          count,      // (in)  #point-parameters
    int          period,    // (in)  subscription period
    PNTNUM*     points,     // (in)  point numbers
    PRMNUM*     params,    // (in)  point parameter numbers
    int*        offsets,   // (in)  point parameter offset
    PARvalue*   values,    // (out) values
    uint2*      types,     // (out) value types
    int*        statuses   // (out) return statuses
);

```

## Arguments

| Argument       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>count</b>   | (in) the number of point parameters in the list to get.                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>period</b>  | (in) subscription period in milliseconds for the point parameters. Use the constant <code>HSC_READ_CACHE</code> if subscription is not required. If the value is in the Experion cache, then that value will be returned. Otherwise the controller will be polled for the latest value. Use the constant <code>HSC_READ_DEVICE</code> if you want to force Experion to poll the controller again. The subscription period will not be applied to standard point types. |
| <b>points</b>  | (in) the list of point numbers.                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>params</b>  | (in) the list of point parameter numbers.                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>offsets</b> | (in) the list of point parameter offsets (for history parameters).                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>values</b>  | <p>(out) the list of values. <b>PARvalue</b> is a union of all possible value data types and is defined as follows (definition from <code>include/src/points.h</code>):</p> <pre> typedef union {     short    int2;     long     int4;     float    real;     double   dble;     char     text[PARAM_MAX_STRING_LEN+1];     struct   {         long   ord;         char   text[PARAM_MAX_STRING_LEN+1];     } en; } PARvalue; </pre>                                  |

| Argument        | Description                                                             |
|-----------------|-------------------------------------------------------------------------|
| <b>types</b>    | the list of value types.                                                |
| <b>statuses</b> | (out) a list containing the status of the get for each point parameter. |

## Description

Retrieves the values for a list of point parameters and stores the values in the **values** union array and returns the data types in **types**.

If your system uses *dynamic scanning*, `hsc_param_values()` calls from the Server API do not trigger dynamic scanning.

## Diagnostics

**0** will be returned from this function upon successful completion, otherwise **-1** will be returned, and calling **c\_geterrno()** will retrieve the error code. The values returned in **types** will be one of the following constants defined in **include\parameters**:

```
DT_CHAR      /* character string                */
DT_INT2      /* 1 to 16 bit short integer                */
DT_INT4      /* 1 to 32 bit long integer                 */
DT_REAL      /* short float                              */
DT_DBLE      /* long float                               */
DT_HIST      /* history (-0 => large float)              */
DT_VAR       /* variant                                  */
DT_ENUM      /* enumeration '                            */
DT_DATE_TIME /* timestamp (integer*2 day, double sec) */
DT_TIME      /* date and time (HSCTIME format)          */
DT_INT8      /* 64-bit integer                          */
DT_SRCADDR   /* source address                           */
DT_DSTADDR   /* destination address                      */
```

---

## Example

Find the values of the Description and PV of 'pntana1' and output them.

```
#include <src/defs.h>
#include <src/points.h>

PNTNUM  points[2];
PRMNUM  params[2];
```

---

---

```
int      offsets[2];
PARvalue values[2];
uint2    types[2];
int      statuses[2];
int      n;

if( (points[0] = hsc_point_number("Pntana1")) == 0
)
{
    printf("pntana1 could not be found!\n");
    return -1;
}

points[1]=points[0];

if( (params[0] = hsc_param_number(points[0],"DESC")) == 0
)
{
    printf("could not find parameter DESC, error code = 0x%x\n", c_
geterrno());
    return -1;
}

if( (params[1] = hsc_param_number(points[1],"PV")) == 0
)
{
    printf("could not find parameter PV, error code = 0x%x\n", c_geterrno
());
    return -1;
}

offsets[0] = offsets[1] = 0;

if( hsc_param_values(2, ONE_SHOT, points, params, offsets, values,
types, statuses) !=0 )
{
    printf("Unable to retrieve parameter, error code = 0x%x\n", c_
```

---

---

```
geterrno());
}
else
{
    for (n=0;n<2;n++)
    {
        if (types[n]==DT_CHAR)
        {
            printf("Point %d param %d is DT_CHAR and the value %s\n", points
[n],params[n],values[n].text);
        }
        else if (types[n]==DT_REAL)
        {
            printf("Point %d param %d is DT_REAL and has value %d\n", points
[n],params[n],values[n].real);
        }
        else
        {
            printf("Unexpected return type %d \n", types[n]);
        }
    }
}
```

---

## See also

*hsc\_param\_value()* on page 167

*hsc\_param\_number()* on page 162

*hsc\_point\_number()* on page 198

## hsc\_param\_value\_put()

Control a point parameter value.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/points.h>
```

```

int hsc_param_value_put(
    PNTNUM        point,
    PRMNUM        param,
    int            offset,
    PARvalue*     value,
    uint2*        type
);

```

## Arguments

| Argument      | Description                                                                                                                                                                                                                                                                                                                                                                         |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>point</b>  | (in) point number                                                                                                                                                                                                                                                                                                                                                                   |
| <b>param</b>  | (in) point parameter number                                                                                                                                                                                                                                                                                                                                                         |
| <b>offset</b> | (in) point parameter offset (for history parameters)                                                                                                                                                                                                                                                                                                                                |
| <b>value</b>  | <p>(in) value. <b>PARvalue</b> is a union of data types and defined as (definition from <b>include/src/points.h</b>):</p> <pre> typedef union {     short    int2;     long     int4;     float    real;     double   dble;     char     text[PARAM_MAX_STRING_LEN+1];     struct {         long    ord;         char    text[PARAM_MAX_STRING_LEN+1];     } en; } PARvalue; </pre> |
| <b>type</b>   | (in) value type                                                                                                                                                                                                                                                                                                                                                                     |

## Description

Sets a value for a point parameter in the server database and performs any control required by setting/changing the parameter's value.

## Diagnostics

On successful write **0** is returned, else an error code is returned. If **CTLOK** (0x8220) is returned this is not actually an error but an indication that some control was executed successfully as a result of setting the parameter value.

## Example

Change Pntana1's SP value to 42.0 and perform any required control.

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/points.h>

PNTNUM   point;
PRMNUM   param;
PARvalue value;
uint2    type;

point = hsc_point_number("Pntana1");
param = hsc_param_number(point,"SP");
value.real = (float)42.0;
type = DT_REAL;
if (hsc_param_value_put(point,param,0,&value,&type) == 0)
    c_logmsg ("example",
              "param_value_put call",
              "Pntana1.SP was written and controlled successfully");
else
    c_logmsg ("example",
              "param_value_put call",
              "Unable to write and/or control Pntana1.SP,
              error code = 0x%x",
              c_geterrno());
```

---

## See also

*hsc\_param\_number()* on page 162

*hsc\_point\_number()* on page 198

*hsc\_param\_values\_put()* below

## **hsc\_param\_values\_put()**

Control a list of point parameter values.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/points.h>

int hsc_param_values_put(
    int count, // (in) number of parameter requests
    PNTNUM* points, // (in) point numbers
    PRMNUM* params, // (in) point parameter numbers
    int* offsets, // (in) point parameter offsets
    PARvalue* values, // (in) values
    uint2* types, // (in) value types
    GDAERR* statuses, // (out) return statuses
    GDASECURITY* security // (in) security descriptor
);
```

## Arguments

| Argument       | Description                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>count</b>   | (in) the number of point parameters in the list to control                                                                                                                                                                                                                                                                                                                                    |
| <b>points</b>  | (in) the list of point numbers                                                                                                                                                                                                                                                                                                                                                                |
| <b>params</b>  | (in) the list of point parameter numbers                                                                                                                                                                                                                                                                                                                                                      |
| <b>offsets</b> | (in) the list of point parameter offsets (for history parameters)                                                                                                                                                                                                                                                                                                                             |
| <b>values</b>  | (in) the list of values. <b>PARvalue</b> is a union of data types and defined as (definition from <b>include/src/points.h</b> ):<br><br><pre>typedef union {     short    int2;     long     int4;     float    real;     double   dble;     char     text[PARAM_MAX_STRING_LEN+1];     struct   {         long  ord;         char  text[PARAM_MAX_STRING_LEN+1];     } en; } PARvalue;</pre> |

| Argument        | Description                                                                                                   |
|-----------------|---------------------------------------------------------------------------------------------------------------|
| <b>types</b>    | (in) the list of value types.                                                                                 |
| <b>statuses</b> | (out) a list containing the status of the put for each point parameter.<br>GDAERR is defined in <hsctypes.h>. |
| <b>security</b> | (in) GDASECURITY is defined in <hsctypes.h>.<br>Use a null pointer for this argument.                         |

## Description

Sets a value for an array of point parameters in the server database and performs any control required by setting/changing the parameter's value.

## Diagnostics

On successful write **0** is returned, else an error code is returned.

The status of each control will be contained in the respective **GDAERR** structure. If **CTLOK** (0x8220) is returned this is not actually an error but an indication that some control was executed successfully as a result of setting the parameter value.

## See also

*hsc\_param\_number()* on page 162

*hsc\_point\_number()* on page 198

*hsc\_param\_value\_put()* on page 174

## hsc\_param\_value\_save()

Save a point parameter value.

## C/C++ synopsis

```
#include <src/defs.h>
```

```
#include <src/points.h>
```

```
int    hsc_param_value_save(
        PNTNUM          point,
        PRMNUM          param,
        int              offset,
        PARvalue*       value,
```

```

        uint2*          type
    );

```

## Arguments

| Argument      | Description                                                                                                                                                                                                                                                                                                                                                                           |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>point</b>  | (in) point number                                                                                                                                                                                                                                                                                                                                                                     |
| <b>param</b>  | (in) point parameter number                                                                                                                                                                                                                                                                                                                                                           |
| <b>offset</b> | (in) point parameter offset (for history parameters)                                                                                                                                                                                                                                                                                                                                  |
| <b>value</b>  | <p>(in) value <b>PARvalue</b> is a union of data types and is defined as (definition from <b>include/src/points.h</b>):</p> <pre> typedef union {     short    int2;     long     int4;     float    real;     double   dble;     char     text[PARAM_MAX_STRING_LEN+1];     struct   {         long   ord;         char   text[PARAM_MAX_STRING_LEN+1];     } en; } PARvalue; </pre> |
| <b>type</b>   | (in) value type                                                                                                                                                                                                                                                                                                                                                                       |

## Description

Sets a value for a point parameter in the server database and does not perform any control which may be required by setting/changing the parameter's value.

## Diagnostics

If the value was written to the parameter correctly then **0** is returned, else **-1** is returned, and calling `c_geterrno()` on page 106 will retrieve the error code.

---

### Example

Change Pntana1's SP value to 42.0.

---

---

```
#include <src/defs.h>
#include <src/points.h>

PNTNUM    point;
PRMNUM    param;
PARvalue  value;
uint2     type;

point = hsc_point_number("Pntana1");
param = hsc_param_number(point,"SP");
value.real = (float)42.0;
type = DT_REAL;
if (hsc_param_value_save(point,param,0,&value,&type) == 0)
    c_logmsg ("example",
              "param_value_save call",
              "Pntana1.SP was written to successfully");
else
    c_logmsg ("example",
              "param_value_save call",
              "Unable to write to Pntana1.SP,
              error code = 0x%x",
              c_geterrno());
```

---

## See also

*hsc\_param\_value\_put()* on page 174

*hsc\_param\_values\_put()* on page 176

## **hsc\_pnttyp\_list\_create()**

List all point types.

## **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/points.h>

int hsc_pnttyp_list_create(
    enumlist**      list
);
```

## Arguments

| Argument    | Description                                                                                                                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>list</b> | <p>(in/out) pointer to an enumeration array for point types. <b>enumlist</b> is defined as (definition from <b>include/src/dictionary.h</b>):</p> <pre>typedef struct {     int    value;     char*  text; } enumlist;</pre> |

## Description

Sets **list** to contain the point types by name and number in the server.

## Diagnostic

The return value of this function will be the number of values in **list** or **-1** if an error occurred. Calling *c\_geterrno()* on page 106 will retrieve the error code.

In all cases the enumlist structure is created with enough space for the text field in each enumlist element in the enumlist array. Because this memory has been allocated by the function, your user code needs to free this space when you finish using the structure. As this function always allocates the memory required for the text field, make sure that you free all memory before calling the routine a second time with the same enumlist\*\* pointer, otherwise there will be a memory leak. To facilitate freeing this memory, *hsc\_enumlist\_destroy()* on page 129 is included in the API.

---

## Example

This code segment uses a list size of 10 and retrieves all point types and outputs their names and numbers, or outputs an error message if the **hsc\_pnttyp\_list\_create()** call was unsuccessful.

```
#include <src/defs.h>
#include <src/points.h>

enumlist* list;
int      i;
int      n;
```

---

---

```
if((i=hsc_pnttyp_list_create
(&list)) != -1)
{
    c_logmsg ("example",
             "pnttyp_list call",
             "The point types available are:");
    for(n=0;n<i;n++)
        c_logmsg ("example",
                 "pnttyp_list call",
                 "%d\t%s",
                 list[n].value,
                 list[n].text);
}
else
    c_logmsg ("example",
             "pnttyp_list call",
             "An error occurred getting point type list,
error code = 0x%x",
             c_geterrno());
```

---

## See also

*hsc\_point\_type()* on page 199

*hsc\_param\_type()* on page 165

*hsc\_enulist\_destroy()* on page 129

## **hsc\_pnttyp\_name()**

Get a point type name.

## **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/points.h>
```

```
int hsc_pnttyp_name(
    int    number,
    char*  name,
```

```

        int    namelen
    );

```

## Arguments

| Argument       | Description                |
|----------------|----------------------------|
| <b>number</b>  | (in) point type number     |
| <b>name</b>    | (out) point type name      |
| <b>namelen</b> | (in) length of name string |

## Description

Returns, in the name char buffer, the name of the specified point type.

## Diagnostics

If the call is successful **0** is returned else **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

## Example

This code segment will retrieve all the point type names with each having their name and number output.

```

#include <src/defs.h>
#include <src/points.h>

int    pnttyp;
char   szPnttyp[10];

pnttyp=1;
while (hsc_pnttyp_name (pnttyp, szPnttyp, 10)
)
    c_logmsg ('example', 'pnttyp_name call',
              'the name of pnttyp %d is %s',
              pnttyp, szPnttyp);
    pnttyp++;
}

```

**See also**

*hsc\_pnttyp\_number()* below

**hsc\_pnttyp\_number()**

Get a point type number.

**C/C++ synopsis**

```
#include <src/defs.h>
#include <src/points.h>

int hsc_pnttyp_number(
    char*    name
);
```

**Arguments**

| Argument | Description          |
|----------|----------------------|
| name     | (in) point type name |

**Description**

Returns the number of the named point type, or if the point type does not exist or an error occurs, **-1** will be returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

**Example**

This code segment should return the point type number for the STA point type and output it, otherwise an error message is output.

```
#include <src/defs.h>
#include <src/points.h>

int    pnttyp;

if((pnttyp = hsc_pnttyp_number('STA'))
    != -1)
    c_logmsg ('example',
              'pnttyp_number call',
```

---

```

        'STA is point type %d',
        pnttyp);
else
    c_logmsg ('example',
             'pnttyp_number call',
             'An error occurred getting point type STA. 0x%x',
             c_geterrno());

```

---

## See also

*hsc\_point\_name()* on page 197

## **hsc\_point\_entityname()**

Returns the entity name of a point.

## C/C++ synopsis

```

#include <src/defs.h>
#include <src/points.h>

int hsc_point_entityname(
    PNTNUM    point,
    char*     name,
    int       namelen
);

```

## Arguments

| Argument       | Description                |
|----------------|----------------------------|
| <b>point</b>   | (in) point number          |
| <b>name</b>    | (out) entity name          |
| <b>namelen</b> | (in) length of name string |

## Description

Takes a point number and returns the point's entity name in the char buffer provided.

## Diagnostic

If successful, **0** is returned. If the point does not exist or some other error occurs, the char buffer is not set and **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

### **hsc\_point\_fullname()**

Returns the full name of the point.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_point_fullname(
    PNTNUM    point,
    char*     name,
    int       namelen
);
```

## Arguments

| Argument       | Description                |
|----------------|----------------------------|
| <b>point</b>   | (in) point number          |
| <b>name</b>    | (out) full name            |
| <b>namelen</b> | (in) length of name string |

## Description

This function takes a point number and return the point's full name in the char buffer provided.

## Diagnostic

If successful, **0** is returned. If the point does not exist or some other error occurs, the char buffer is not set and **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

### **hsc\_point\_get\_children()**

Returns all children.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_point_get_children(
    PNTNUM    point,
    int*      count,
    PNTNUM**  children
);
```

## Arguments

| Argument        | Description              |
|-----------------|--------------------------|
| <b>point</b>    | (in) point number        |
| <b>count</b>    | (out) number of children |
| <b>children</b> | (out) array of children  |

## Description

This function returns all children, both containment and reference.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```
#include <src/defs.h>
#include <src/points.h>

int count = 0;
PNTNUM *Children = NULL;
if (hsc_point_get_children (point, &count, &Children) != 0)
    return -1
.
.
```

---

---

```
hsc_em_FreePointList (Children);
```

---

## hsc\_point\_get\_containment\_ancestors()

Returns all containment ancestors above a specified point.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_point_get_containment_ancestors (
    PNTNUM    point,
    int*      piNumAncestors,
    PNTNUM**  ppAncestors
);
```

### Arguments

| Argument              | Description               |
|-----------------------|---------------------------|
| <b>point</b>          | (in) point number         |
| <b>piNumAncestors</b> | (out) number of ancestors |
| <b>ppAncestors</b>    | (out) array of ancestors  |

### Description

This function returns all of the containment ancestors in the tree above the specified point.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

### Diagnostics

If successful, **0** is returned, otherwise **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

**Example**

```

#include <src/defs.h>
#include <src/points.h>

int iNumAncestors = 0;
PNTNUM *pAncestors = NULL
if (hsc_point_get_containment_ancestors (point, &iNumAncestors,
&pAncestors) != 0)
    return -1
.
.
.
hsc_em_FreePointList (pAncestors);

```

---

**hsc\_point\_get\_containment\_children()**

Returns all containment children for a specified point.

**C/C++ synopsis**

```

#include <src/defs.h>
#include <src/points.h>

int hsc_point_get_containment_children(
    PNTNUM    parent,
    int*      piNumChildren,
    PNTNUM**  ppChildren
);

```

**Arguments**

| Argument             | Description                     |
|----------------------|---------------------------------|
| <b>parent</b>        | (in) point number of the parent |
| <b>piNumChildren</b> | (out) number of children        |
| <b>ppChildren</b>    | (out) array of children         |

**Description**

Returns a list of contained children for a specified point.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```
#include <src/defs.h>
#include <src/points.h>

int iNumChildren = 0;
PNTNUM *pChildren = NULL
if (hsc_point_get_containment_children (point, &iNumChildren, &pChildren) != 0)
    return -1
.
.
.
hsc_em_FreePointList (pChildren);
```

---

## **hsc\_point\_get\_containment\_descendants()**

Returns all containment descendants below a specified point.

### **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/points.h>

int hsc_point_get_containment_descendants (
    PNTNUM    point,
    int*      piNumDescendants,
    PNTNUM**  ppDescendants
);
```

## Arguments

| Argument                | Description                 |
|-------------------------|-----------------------------|
| <b>point</b>            | (in) point number           |
| <b>piNumDescendents</b> | (out) number of descendents |
| <b>ppDescendents</b>    | (out) array of descendents  |

## Description

Returns a list of containment descendents in the tree below a specified point.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```
#include <src/defs.h>
#include <src/points.h>

int iNumDescendents = 0;
PNTNUM *pDescendents = NULL;
if (hsc_point_get_containment_descendents (point, &iNumDescendents,
&pDescendents) != 0)
    return -1
.
.
.
hsc_em_FreePointList (pDescendents);
```

---

## **hsc\_point\_get\_containment\_parents()**

Returns all containment parents above a specified point.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_point_get_containment_parents(
    PNTNUM    child,
    int*      piNumParents,
    PNTNUM**  ppParents
);
```

## Arguments

| Argument            | Description                          |
|---------------------|--------------------------------------|
| <b>child</b>        | (in) point number of the child point |
| <b>piNumParents</b> | (out) number of parents              |
| <b>ppParents</b>    | (out) array of parents               |

## Description

Returns a list of containment parents for a specified point.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```
#include <src/defs.h>
#include <src/points.h>

int iNumParents = 0;
PNTNUM *pParents = NULL;
if (hsc_point_get_containment_parents (point, &iNumParents, &pParents)
    != 0)
    return -1
.
```

---

---

```

.
.
hsc_em_FreePointList (pParents);

```

---

## **hsc\_point\_get\_parents()**

Returns all parents.

### **C/C++ synopsis**

```

#include <src/defs.h>
#include <src/points.h>

```

```

int hsc_point_get_parents(
    PNTNUM    point,
    int*      count,
    PNTNUM**  parents
);

```

### **Arguments**

| <b>Argument</b> | <b>Description</b>      |
|-----------------|-------------------------|
| <b>point</b>    | (in) point number       |
| <b>count</b>    | (out) number of parents |
| <b>parents</b>  | (out) array of parents  |

### **Description**

Returns all parents, both containment and reference.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

### **Diagnostics**

If successful, **0** is returned, otherwise **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

**Example**

```

#include <src/defs.h>
#include <src/points.h>

int count = 0;
PNTNUM *Parents = NULL
if (hsc_point_get_parents (point, &count, &Parents) != 0)
    return -1
.
.
.
hsc_em_FreePointList (Parents);

```

---

**hsc\_point\_get\_references()**

Returns a list of points to which the specified point refers.

**C/C++ synopsis**

```

#include <src/defs.h>
#include <src/points.h>

int hsc_point_get_references (
    PNTNUM    point,
    int*      piNumRefItems,
    PNTNUM**  ppRefItems
);

```

**Arguments**

| Argument             | Description                     |
|----------------------|---------------------------------|
| <b>point</b>         | (in) point number               |
| <b>piNumRefItems</b> | (out) number of references      |
| <b>ppRefItems</b>    | (out) array of referenced items |

**Description**

Returns a list of points to which the specified point refers.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```
#include <src/defs.h>
#include <src/points.h>

int iNumRefItems = 0;
PNTNUM *pRefItems = NULL
if (hsc_point_get_references (point, &iNumRefItems, &pRefItems) != 0)
    return -1
.
.
.
hsc_em_FreePointList (pRefItems);
```

---

## hsc\_point\_get\_referers()

Returns a list of points that refer to the specified point.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_point_get_referers (
    PNTNUM    point,
    int*      piNumRefItems,
    PNTNUM**  ppRefItems
);
```

### Arguments

| Argument     | Description       |
|--------------|-------------------|
| <b>point</b> | (in) point number |

| Argument             | Description                     |
|----------------------|---------------------------------|
| <b>piNumRefItems</b> | (out) number of referers        |
| <b>ppRefItems</b>    | (out) array of referring points |

## Description

Returns a list of points that refer to the specified point.

The array must be cleared by calling *hsc\_em\_FreePointList()* on page 124.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```
#include <src/defs.h>
#include <src/points.h>

int iNumRefItems = 0;
PNTNUM *pRefItems = NULL;
if (hsc_point_get_referers (point, &iNumRefItems, &pRefItems) != 0)
    return -1
.
.
.
hsc_em_FreePointList (pRefItems);
```

---

## **hsc\_point\_guid()**

Returns the GUID for the specified point.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

int hsc_point_guid(
    PNTNUM    point,
```

```

        GUID*    pGUID
    );

```

## Arguments

| Argument     | Description                 |
|--------------|-----------------------------|
| <b>point</b> | (in) point number           |
| <b>gGUID</b> | (out) GUID in binary format |

## Description

Returns the **GUID** for the specified point in binary format.

## Diagnostics

If successful, **0** is returned, otherwise **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

## hsc\_point\_name()

Gets a point's name.

## C/C++ synopsis

```

#include <src/defs.h>
#include <src/points.h>

int hsc_point_name (
    PNTNUM    point,
    char*     name,
    int       namelen
);

```

## Arguments

| Argument       | Description                |
|----------------|----------------------------|
| <b>point</b>   | (in) point number          |
| <b>name</b>    | (out) parameter name       |
| <b>namelen</b> | (in) length of name string |

## Description

This routine will take a point number and return the point's name in the char buffer provided and have a return value of **0**.

## Diagnostic

If the point does not exist or some other error occurs then the char buffer will not be set and **-1** will be returned. The error code can be retrieved by calling *c\_geterrno()* on page 106.

## Example

Retrieves the point name for the point and outputs it. The char buffer is 41 characters long, which is bigger than the maximum length of a point name (40 characters).

```
#include <src/defs.h>
#include <src/points.h>

PNTNUM point;
char    szName[MAX_POINT_NAME_LEN+1];

if (hsc_point_name (point, szName, MAX_POINT_NAME_LEN+1) == 0)
    c_logmsg ('example',
              'point_name call',
              'Point %d is named %s',
              point,
              szName);
else
    c_logmsg ('example',
              'point_name call',
              'An error occurred getting point name. 0x%x',
              c_geterrno());
```

## See also

*hsc\_point\_number()* below

## **hsc\_point\_number()**

Gets a point's number.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

PNTNUM hsc_point_number(
    char*    name
);
```

## Arguments

| Argument | Description     |
|----------|-----------------|
| name     | (in) point name |

## Description

This function returns the point number of the given point name if the point exists else **0** is returned.

### Example

The point number is retrieved for the point and output, if the point exists then a message is output.

```
#include <src/defs.h>
#include <src/points.h>
PNTNUM point;

if((point = hsc_point_number('pntana1')) !=0)
    c_logmsg ('example','param_number call',
        'pntana1 is point number %d',point);
```

## See also

*hsc\_point\_name()* on page 197

## **hsc\_point\_type()**

Gets a point's type.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

PNTTYP hsc_point_type(
    PNTNUM    point
);
```

## Argument

| Argument     | Description       |
|--------------|-------------------|
| <b>point</b> | (in) point number |

## Description

This routine returns the point type of the point and will be one of the following values (as defined in **include\parameters**) or **-1** if invalid:

```
#define STA 1
#define ANA 2
#define ACC 3
#define ACS 4
#define CON 5
#define CDA 6
#define RDA 7
#define PSA 8
```

---

## Example

Calculates the point number from the point name and then uses this value to determine the point type. The point number and type are then output.

```
#include <src/defs.h>
#include <src/points.h>

PNTNUM point;
PNTTYP pnttyp;

point = hsc_point_number('PNTANA1');
if (point != 0)
```

---

---

```

{
    pnttyp = hsc_point_type(point);
    c_logmsg ('example', 'point_type call',
             'PNTANA1 is point number %d has point type %d.',
             point, pnttyp);
}

```

---

## hsc\_StringFromGUID()

Converts a **GUID** from binary format to string format.

### C/C++ synopsis

```

#include <src/defs.h>
#include <src/points.h>

int hsc_StringFromGUID(
    GUID*    pGUID,
    char*    pszGUID
);

```

### Arguments

| Argument | Description                 |
|----------|-----------------------------|
| pGUID    | (in) GUID in binary format  |
| pszGUID  | (out) GUID in string format |

### Description

This function converts a **GUID** from binary format to string format.

### Diagnostics

If successful, **0** is returned, otherwise **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve the error code.

---

### Example

```
#include <src/defs.h>
#include <src/points.h>

GUID guid;
char szGUID[MAX_GUID_STRING_SZ+1];
hsc_StringFromGUID (&guid, szGUID);
if (point == 0)
    return -1;
```

---

### hsc\_unlock\_file()

Unlocks a database file.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int hsc_unlock_file(
    int file
);
```

### Arguments

| Argument     | Description                                                    |
|--------------|----------------------------------------------------------------|
| <b>file</b>  | (in) server file number.                                       |
| <b>delay</b> | (in) delay time in milliseconds before lock attempt will fail. |

### Description

Performs advisory unlocking of database files. Advisory locking means that the tasks that use the file take responsibility for setting and removing locks as needed.

For more information regarding database locking see 'Ensuring database consistency.'

### Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                 |                                         |
|-----------------|-----------------------------------------|
| <b>[FILLCK]</b> | File locked to another task             |
| <b>[RECLCK]</b> | Record locked to another task           |
| <b>[DIRLCK]</b> | File's directory locked to another task |
| <b>[BADFIL]</b> | Illegal file number specified           |

## See also

*hsc\_lock\_record()* on page 145

*hsc\_unlock\_file()* on the previous page

*hsc\_unlock\_record()* below

## **hsc\_unlock\_record()**

Unlocks a record of a database file.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>
```

```
int hsc_unlock_record(
    int file,
    int record
);
```

## Arguments

| Argument      | Description                          |
|---------------|--------------------------------------|
| <b>file</b>   | (in) server file number              |
| <b>record</b> | (in) record number (see description) |

## Description

This routine is used to perform advisory unlocking of database files and records. Advisory locking means that the tasks that use the record take responsibility for setting and removing locks as needed.

For more information regarding database locking see 'Ensuring database consistency.'

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                  |                                         |
|------------------|-----------------------------------------|
| <b>[FILLCK]</b>  | File locked to another task             |
| <b>[RECLCK]</b>  | Record locked to another task           |
| <b>[DIRLCK]</b>  | File's directory locked to another task |
| <b>[BADFIL]</b>  | Illegal file number specified           |
| <b>[BADRECD]</b> | Illegal record number specified         |

## See also

*hsc\_lock\_file()* on page 144

*hsc\_unlock\_file()* on page 202

*hsc\_unlock\_record()* on the previous page

## HsctimeToDate()

Converts date/time stored in **HSCTIME** format to **VARIANT DATE** format.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

void HsctimeToDate(
    HSCTIME*,
    DATE*
);
```

## Description

Converts a date/time value stored in **HSCTIME** format to **VARIANT DATE** format.

## HsctimeToFiletime()

Converts date/time stored in **HSCTIME** format to **FILETIME** format.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/points.h>

void HsctimeToFiletime (
    HSCTIME*,
    FILETIME*
);
```

## Description

Converts a date/time value stored in **HSCTIME** format to **FILETIME** format.

## infdouble()

Returns the (IEEE 754) **Infinity** value as a **Double** data type.

## C/C++ Synopsis

```
double infdouble();
```

## Arguments

No arguments are passed with this function.

## Returns

Returns the (IEEE 754) positive **Infinity** value as a **Double** (double-precision floating-point) data type.

## Description

This function is useful for creating a valid (IEEE 754) **Infinity** value for comparison purposes with **Double** data type variables.

---

### Attention:

This function is platform dependent (only valid for an INTEL X86 system).

---

## See also

*Validating IEEE 754 special values* on page 20

*isinfdouble()* on page 212

*inffloat()* below

*isinffloat()* on page 213

*Data types* on page 26

## **inffloat()**

Returns the (IEEE 754) **Infinity** value as a **Float** data type.

### **C/C++ Synopsis**

```
float inffloat();
```

### **Arguments**

No arguments are passed with this function.

### **Returns**

Returns the (IEEE 754) positive **Infinity** value as a **Float** (single-precision floating-point) data type.

### **Description**

This function is useful for creating a valid (IEEE 754) **Infinity** value for comparison purposes with **Float** data type variables.

---

#### **Attention:**

This function is platform dependent (only valid for an INTEL X86 system).

---

### **See also**

*Validating IEEE 754 special values* on page 20

*isinffloat()* on page 213

*infdouble()* on the previous page

*isindouble()* on page 212

*Data types* on page 26

## **Int2toPV()**

Inserts an **int2** value into a **PARvalue** union.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/almmsg.h>

PARvalue* Int2toPV(
    int2          int2_val,
    PARvalue*    pvvalue
);
```

## Arguments

| Argument        | Description                                                              |
|-----------------|--------------------------------------------------------------------------|
| <b>pvvalue</b>  | (in) A pointer to a <b>PARvalue</b> structure.                           |
| <b>int2_val</b> | (in) The <b>int2</b> value to insert into the <b>PARvalue</b> structure. |

## Description

This function inserts an **int2** value into a **PARvalue**, and then returns a pointer to the **PARvalue** passed in. This function allows you to set attributes into a notification structure using calls to *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138 functions in a single line of code.

## Diagnostics

If this function is successful, the return value is a pointer back to the **PARvalue** passed in, otherwise, the return value is **NULL** and calling *c\_geterrno()* on page 106 will retrieve the following error code:

|                         |                                                               |
|-------------------------|---------------------------------------------------------------|
| <b>BUFFER_TOO_SMALL</b> | The pointer to the <b>PARvalue</b> is invalid, that is, null. |
|-------------------------|---------------------------------------------------------------|

## Example

See the examples in *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138.

## See also

*DbletoPV()* on page 98

*hsc\_insert\_attrib()* on page 131

*hsc\_insert\_attrib\_byindex()* on page 138

*Int4toPV()* below

*PritoPV()* on page 228

*RealtoPV()* on page 231

*StrtoPV()* on page 240

*TimetoPV()* on page 242

## Int4toPV()

Inserts an **int4** value into a **PARvalue** union.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/almmsg.h>

PARvalue* Int4toPV(
    int4          int4_val,
    PARvalue*    pvvalue
);
```

### Arguments

| Argument        | Description                                                              |
|-----------------|--------------------------------------------------------------------------|
| <b>pvvalue</b>  | (in) A pointer to a <b>PARvalue</b> structure.                           |
| <b>int4_val</b> | (in) The <b>int4</b> value to insert into the <b>PARvalue</b> structure. |

### Description

This function inserts an **int4** value into a **PARvalue**, and then returns a pointer to the **PARvalue** passed in. This function allows you to set attributes into a notification structure using calls to *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138 functions in a single line of code.

### Diagnostics

If this function is successful, the return value is a pointer back to the **PARvalue** passed in, otherwise, the return value is **NULL** and calling *c\_geterrno()* on page 106 will retrieve the following error code:

|                         |                                                               |
|-------------------------|---------------------------------------------------------------|
| <b>BUFFER_TOO_SMALL</b> | The pointer to the <b>PARvalue</b> is invalid, that is, null. |
|-------------------------|---------------------------------------------------------------|

---

**Example**

See the examples in *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138.

---

**See also**

*DbletoPV()* on page 98

*hsc\_insert\_attrib()* on page 131

*hsc\_insert\_attrib\_byindex()* on page 138

*Int2toPV()* on page 206

*PritoPV()* on page 228

*RealtoPV()* on page 231

*StrtoPV()* on page 240

*TimetoPV()* on page 242

**c\_intchr()**

Copies an integer array to a character string.

**C/C++ synopsis**

```
#include <src/defs.h>
```

```
void __stdcall c_intchr(  
    int2*    intbuf,  
    int      intbuflen,  
    char*    chrbuf,  
    int      chrbuflen  
);
```

**Arguments**

| Argument         | Description                                  |
|------------------|----------------------------------------------|
| <b>intbuf</b>    | (in) source integer buffer containing ASCII. |
| <b>intbuflen</b> | (in) size of source integer buffer in bytes. |

| Argument         | Description                                                                              |
|------------------|------------------------------------------------------------------------------------------|
| <b>chrbuf</b>    | (out) destination character buffer.                                                      |
| <b>chrbuflen</b> | (in) size of character buffer in bytes (to allow non null-terminated character buffers). |

## Description

Copies characters from an integer buffer into a character buffer. It will either space fill or truncate so as to ensure that **chrbuflen** characters are copied into the character buffer. If the system stores words with the least significant byte first then byte swapping will be performed with the move.

## See also

*c\_chrint()* on page 85

## IsGDAerror()

Determines whether a returned **GDA** status value is an error.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/gdamacro.h>

bool IsGDAerror
(
    GDAERR* pGdaError
);
```

## Arguments

| Argument         | Description                                                |
|------------------|------------------------------------------------------------|
| <b>pGdaError</b> | (in) pointer to the DGAERR structure containing the status |

## Description

Determines whether a particular **GDA** status code is an error by returning a 0 in the return value. If the function returns a non zero value, calling *c\_geterrno()* on page 106 will retrieve the error code.

## Diagnostics

Returns **TRUE** if **pGdaError** indicates an error condition, otherwise it returns **FALSE**.

**See also***IsGDAwarning()* on the next page*IsGDAnoerror()* below*hsc\_IsError()* on page 142*c\_geterrno()* on page 106**IsGDAnoerror()**

Determines whether a returned **GDA** status value is neither an error nor a warning.

**C/C++ synopsis**

```
#include <src/defs.h>
#include <src/gdamacro.h>

bool IsGDAnoerror (
    GDAERR* pGdaError
);
```

**Arguments**

| Argument         | Description                                                |
|------------------|------------------------------------------------------------|
| <b>pGdaError</b> | (in) pointer to the GDAERR structure containing the status |

**Description**

This macro determines whether a particular **GDA** status code is an error by returning a 0 in the return value. If the function returns a non zero value, calling *c\_geterrno()* on page 106 will retrieve the error code.

**Diagnostics**

This routine returns **TRUE** if **pGdaError** indicates neither an error condition nor a warning condition, otherwise it returns **FALSE**.

**See also***IsGDAwarning()* on the next page*IsGDAerror()* on the previous page*c\_geterrno()* on page 106

## IsGDAwarning()

Determines whether a returned **GDA** status value is a warning.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/gdamacro.h>
```

```
bool IsGDAwarning(
    GDAERR* pGdaError
);
```

### Arguments

| Argument         | Description                                                       |
|------------------|-------------------------------------------------------------------|
| <b>pGdaError</b> | (in) pointer to the <b>GDAERR</b> structure containing the status |

### Description

This macro determines whether a particular **GDA** status code is an error by returning a 0 in the return value. If the function returns a non zero value, calling *c\_geterrno()* on page 106 will retrieve the error code.

### Diagnostics

This routine returns **TRUE** if **pGdaError** indicates a warning condition, otherwise it returns **FALSE**.

### See also

*IsGDAerror()* on page 210

*IsGDAnoerror()* on the previous page

*hsc\_IsWarning()* on page 143

*c\_geterrno()* on page 106

## isinfdouble()

Validates the Double data type argument as an (IEEE 754) **Infinity** value.

### C/C++ Synopsis

```
int isinfdouble(double ieee);
```

## Arguments

| Argument | Description                      |
|----------|----------------------------------|
| ieee     | (in) An (IEEE 754) value to test |

## Returns

Returns **TRUE (-1)** if it is a valid (IEEE 754) **Infinity** value; otherwise **FALSE (0)**.

## Description

This function is useful for checking that the argument is, or is not, a valid (IEEE 754) **Infinity** value.

---

### Attention:

This function is platform dependent (only valid for an INTEL X86 system).

---

## See also

*Validating IEEE 754 special values* on page 20

*infdouble()* on page 205

*inffloat()* on page 206

*isinffloat()* below

*Data types* on page 26

## isinffloat()

Validates the **Float** data type argument as an (IEEE 754) **Infinity** value.

## C/C++ Synopsis

```
int isinffloat(float ieee);
```

## Arguments

| Argument | Description                      |
|----------|----------------------------------|
| ieee     | (in) An (IEEE 754) value to test |

## Returns

Returns **TRUE (-1)** if it is a valid (IEEE 754) **Infinity** value; otherwise **FALSE (0)**.

## Description

This function is useful for checking that the argument is, or is not, a valid (IEEE 754) **Infinity** value.

---

### Attention:

Note that this function is platform dependent (only valid for an INTEL X86 system).

---

## See also

*Validating IEEE 754 special values* on page 20

*inffloat()* on page 206

*infdouble()* on page 205

*isinfdouble()* on page 212

*Data types* on page 26

## isnandouble()

Validates the Double data type argument as an (IEEE 754) **NaN** (Not a Number) value.

## C/C++ Synopsis

```
int isnandouble(double ieee);
```

## Arguments

| Argument          | Description                      |
|-------------------|----------------------------------|
| <code>ieee</code> | (in) An (IEEE 754) value to test |

## Returns

Returns **TRUE (-1)** if it is a valid (IEEE 754) **NaN** value; otherwise **FALSE (0)**.

## Description

This function is useful for checking that the argument is, or is not, a valid (IEEE 754) **NaN** value. For example, a controller can send an IEEE 754 NaN value in response to a data request, which should be tested for.

---

### Attention:

This function is platform dependent (only valid for an INTEL X86 system).

---

## See also

*Validating IEEE 754 special values* on page 20

*nandouble()* on page 219

*nanfloat()* on page 219

*isnanfloat()* below

*Data types* on page 26

## isnanfloat()

Validates the **Float** data type argument as an (IEEE 754) **NaN** (Not a Number) value.

## C/C++ Synopsis

```
int isnanfloat(float ieee);
```

## Arguments

| Argument | Description                      |
|----------|----------------------------------|
| ieee     | (in) An (IEEE 754) value to test |

## Returns

Returns **TRUE** (-1) if it is a valid (IEEE 754) **NaN** value; otherwise **FALSE** (0).

## Description

This function is useful for checking that the argument is, or is not, a valid (IEEE 754) **NaN** value. For example, a controller can send an IEEE 754 NaN value in response to a data request, which should be tested for.

**Attention:**

This function is platform dependent (only valid for an INTEL X86 system).

**See also**

*Validating IEEE 754 special values* on page 20

*nanfloat()* on page 219

*nandouble()* on page 219

*isnandouble()* on page 214

*Data types* on page 26

**Julian/Gregorian date conversion()**

Convert between Julian days and Gregorian date.

**C/C++ synopsis**

```
#include <src/defs.h>

int __stdcall c_gtoj (
    int    year,
    int    month,
    int    day
);

void __stdcall c_jtog(
    int    julian,
    int*   year,
    int*   month,
    int*   day
);
```

**Arguments**

| Argument     | Description                                              |
|--------------|----------------------------------------------------------|
| <b>year</b>  | (in/out) number of years since 0 AD (for example, 2012). |
| <b>month</b> | (in/out) month (1–12).                                   |

| Argument      | Description                 |
|---------------|-----------------------------|
| <b>day</b>    | (in/out) day (1–31).        |
| <b>julian</b> | (in) number of Julian days. |

## Description

Converts between Julian days (used by the History subsystem) and a Gregorian date (day, month, year format).

|               |                                                |
|---------------|------------------------------------------------|
| <b>c_gtoj</b> | converts from a Gregorian date to Julian days. |
| <b>c_jtog</b> | converts from Julian days to a Gregorian date. |

## Diagnostics

Upon successful completion **c\_gtoj** returns the number of Julian days. **c\_jtog** returns the values in the addresses pointed to by the year, month and day parameters.

## See also

*c\_gethstpar...\_2()* on page 107

## c\_logmsg()

Writes a message to the log file.

## C/C++ synopsis

```
#include <src/defs.h>

void __stdcall c_logmsg(
    char*    progname,
    char*    lineno,
    char*    format,
    ...
);
```

## Arguments

| Argument        | Description                        |
|-----------------|------------------------------------|
| <b>progname</b> | (in) name of program module        |
| <b>lineno</b>   | (in) line number in program module |

| Argument      | Description                        |
|---------------|------------------------------------|
| <b>format</b> | (in) printf type format of message |

## Description

**c\_logmsg** should be used instead of **printf** to write messages to the log file. This is typically used for debugging purposes.

This routine writes the message to standard error. If the application is a task (with its own LRN) then the message will be captured and written to the log file.

If the program is a utility then the message will appear in the command prompt window.

## Diagnostics

This routine has no return value. If it is called incorrectly it will write its own message to standard error indicating the source of the problem.

### Example

```
c_logmsg('abproc.c', '134' 'Point ABSTAT001 PV out of normal range
(%d)', abpv);
```

**c\_logmsg** handles all carriage control. There is no need to put a line feed characters in calls to **c\_logmsg**.

## c\_mzero()

Tests a real value for -0.0.

## C/C++ synopsis

```
#include <src/defs.h>

int __stdcall c_mzero(
    float* value
);
```

## Arguments

| Argument     | Description                   |
|--------------|-------------------------------|
| <b>value</b> | (in) real value to be tested. |

## Description

Returns **TRUE** if the specified value is equal to minus zero. Otherwise, **FALSE** is returned. Minus zero is used to represent bad data in history data.

## nandouble()

Returns the (IEEE 754) **QNaN** (Quiet Not a Number) value as a Double data type.

## C/C++ Synopsis

```
double nandouble();
```

## Arguments

No arguments are passed with this function.

## Returns

Returns the (IEEE 754) **QNaN** value as a Double (double-precision floating-point) data type.

## Description

This function is useful for creating a valid (IEEE 754) **QNaN** value for storage.

---

### Attention:

This function is platform dependent (only valid for an INTEL X86 system).

---

## See also

*Validating IEEE 754 special values* on page 20

*isnandouble()* on page 214

*nanfloat()* below

*isnanfloat()* on page 215

*Data types* on page 26

## nanfloat()

Returns the (IEEE 754) **QNaN** (Quiet Not a Number) value as a **Float** data type.

## C/C++ Synopsis

```
float nanfloat();
```

## Arguments

No arguments are passed with this function.

## Returns

Returns the (IEEE 754) **QNaN** value as a **Float** (single-precision floating-point) data type.

## Description

This function is useful for creating a valid (IEEE 754) **QNaN** value for storage.

---

### Attention:

This function is platform dependent (only valid for an INTEL X86 system).

---

## See also

*Validating IEEE 754 special values* on page 20

*isnanfloat()* on page 215

*nandouble()* on the previous page

*isnandouble()* on page 214

*Data types* on page 26

## c\_oprstr\_...()

Sends a message to a Station.

## C/C++ synopsis

```
#include <system>
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/trbtbl_def>
```

```
// Select one of the following synopses as appropriate to
```

```
// the type of message being sent

int  __stdcall c_oprstr_info(
    int      crt,
    char*    message
);

int  __stdcall c_oprstr_message(
    int      crt,
    char*    message
);

int  __stdcall c_oprstr_prompt(
    int      crt,
    char*    message,
    int      param1
);

char * __stdcall c_oprstr_response(
    int      crt,
    struct prm* prmbk
);
```

## Arguments

| Argument       | Description                                                                                      |
|----------------|--------------------------------------------------------------------------------------------------|
| <b>crt</b>     | (in) Station number.                                                                             |
| <b>message</b> | (in) pointer to null-terminated string to be sent to the Station.                                |
| <b>param1</b>  | (in) value of parameter 1 required to identify response task request.                            |
| <b>prmbk</b>   | (out) task request parameter block which was received from GETREQ when the task began executing. |

## Description

Outputs a specified message to the Operator zone of a Station.

|                      |                                                                                  |
|----------------------|----------------------------------------------------------------------------------|
| <b>c_oprstr_info</b> | outputs information only messages.                                               |
| <b>c_</b>            | outputs an invalid request message that is cleared after a TIME_REQUEST seconds. |

|                          |                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>oprstr_message</b>    | This constant is defined in 'server/src/system'.                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>c_oprstr_prompt</b>   | outputs an operator prompt and sets the ENTER key to notify the calling task with the specified parameter 1. After calling this routine the task should branch back to its GETREQ call to service its next request, and if none exists, then terminate with a TRM04. When the operator types a response and presses ENTER, the task is requested with the specified parameter 1, and should branch to code that calls <code>c_oprstr_response</code> to fetch the response. |
| <b>c_oprstr_response</b> | reads the entered data and clears the prompt.                                                                                                                                                                                                                                                                                                                                                                                                                               |

## Diagnostics

Upon successful completion `c_oprstr_response` will return a pointer to a null-terminated string containing the response from the operator. Upon successful completion `c_oprstr_info`, `c_oprstr_message` and `c_oprstr_prompt` will return zero. Otherwise, a **NULL** pointer or **-1** will be returned, and calling `c_geterrno()` on page 106 will retrieve the following error code:

|                         |                               |
|-------------------------|-------------------------------|
| <b>[M4_INVALID_CRT]</b> | An invalid CRT was specified. |
|-------------------------|-------------------------------|

## Warnings

`c_oprstr_prompt` requires that the calling task `c_trm04()` on page 245 until the ENTER key is pressed. This means that the calling task must be a server task and not a utility task.

If the operator changes displays or presses the ESC key after a `c_oprstr_prompt` call, no operator input will be saved, and the task will not be requested. If you do not want the operator to avoid entering data, you will need to set a flag internal to your program that indicates whether a response has been received and start a task timer with the `c_tmstrt_...()` on page 244 function. If a response has not been received from the operator after an interval (specified by you in the `c_tmstrt_...()` on page 244 function), you will need to repeat the `c_oprstr_prompt` call. If a response from the operator is received you will need to stop the timer using `c_tmstop()` on page 243.

---

### Example

```
#include <stdio.h> /* for NULL */
#include "src/defs.h"
#include "src/M4_err.h"
#include "src/trbtbl_def"
```

---

---

```
static progname="%M%";
main ()
{
    struct prm prmblk;
    int crt=1;
    char *reply;
    uint2 point;
        ...
        ...
        ...
    if (c_getreq(&prmblk) == 0)
    {
        switch (prmblk.param1)
        {
        case 1:
            /* request a point name from the operator */
            if (c_oprstr_prompt(crt,"ENTER POINT NAME?",2))
                c_logmsg (progname,"123","c_oprstr_prompt %x",c_geterrno());
            break;
        case 2:
            reply=c_oprstr_response(crt,&prmblk);
            if (reply==NULL)
                c_logmsg(progname,"132",
                    "c_oprstr_response error %x",c_geterrno());
            else
            {
                if (c_getpnt(reply,point) == -1)
                    c_oprstr_message(crt,"ILLEGAL POINT NAME");
            }
            else
            {
                /* we have a valid point type/number */
            }
            }
            break;
        } /* end switch */
    }
}
```

---

## **c\_pps\_2()**

Processes a point special (without waiting for completion).

## C/C++ synopsis

```
#include <src/defs.h>
#include <scr/M4_err.h>

int __stdcall c_pps_2
(
    PNTNUM    point,
    PRMNUM    param,
    int*      status
);
```

## Arguments

| Argument      | Description                                                  |
|---------------|--------------------------------------------------------------|
| <b>point</b>  | (in) Point type/number to be processed.                      |
| <b>param</b>  | (in) Point parameter to be processed. -1 for all parameters. |
| <b>status</b> | (out) return error code.                                     |

## Description

Requests a demand scan of the specified point.

The point is always processed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion, a value of **0** is returned in status. Otherwise, one of the following error codes is returned:

|                     |                                                            |
|---------------------|------------------------------------------------------------|
| [M4_INV_POINT]      | Invalid point specified                                    |
| [MR_INV_PARAMETER]  | Invalid point parameter specified                          |
| [M4_DEVICE_TIMEOUT] | Scan was not performed before a 10 second timeout interval |

## See also

*c\_ppsw\_2()* on the next page

## c\_ppsw\_2()

Processes a point special and waits for completion.

### C/C++ synopsis

```
#include <src/defs.h>
#include <scr/M4_err.h>

int __stdcall c_ppsw_2
(
    PNTNUM    point,
    PRMNUM    param,
    int*      status
);
```

### Arguments

| Argument      | Description                                                  |
|---------------|--------------------------------------------------------------|
| <b>point</b>  | (in) point type/number to be processed.                      |
| <b>param</b>  | (in) point parameter to be processed. -1 for all parameters. |
| <b>status</b> | (out) return error code.                                     |

### Description

Requests a demand scan of the specified point and wait for the scan to complete. If, after 10 seconds, the scan has not replied, a timeout will be indicated.

The point is always processed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

### Diagnostics

Upon successful completion, a value of **0** is returned in status. Otherwise, one of the following error codes is returned:

|                     |                                                             |
|---------------------|-------------------------------------------------------------|
| [M4_INV_POINT}      | Invalid point specified.                                    |
| [M4_INV_PARAMETER]  | Invalid point parameter specified.                          |
| [M4_DEVICE_TIMEOUT] | Scan was not performed before a 10 second timeout interval. |

## See also

*c\_pps\_2()* on page 223

## c\_ppv\_2()

Processes a point value (without waiting for completion).

## C/C++ synopsis

```
#include <src/defs.h>
#include <scr/M4_err.h>

int __stdcall c_ppv_2
(
    PNTNUM    point,
    PRMNUM    param,
    float     value
);
```

## Arguments

| Argument     | Description                                                  |
|--------------|--------------------------------------------------------------|
| <b>point</b> | (in) Point type/number to be processed.                      |
| <b>param</b> | (in) Point parameter to be processed. -1 for all parameters. |
| <b>value</b> | (in) Value to be stored into the point parameter.            |

## Description

Requests a Demand scan of the specified Point.

The point is always processed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                       |                          |
|-----------------------|--------------------------|
| <b>[M4_INV_POINT]</b> | Invalid Point specified. |
|-----------------------|--------------------------|

|                            |                                                                     |
|----------------------------|---------------------------------------------------------------------|
| <b>[M4_INV_PARAMETER]</b>  | Invalid Point parameter specified.                                  |
| <b>[M4_ILLEGAL_RTU]</b>    | There are no Controllers implemented that can process this request. |
| <b>[M4_DEVICE_TIMEOUT]</b> | Scan was not performed before a 10 second timeout interval.         |

## See also

*c\_ppvw\_2()* below

## **c\_ppvw\_2()**

Processes a point value and waits for completion.

## C/C++ synopsis

```
#include <src/defs.h>
#include <scr/M4_err.h>

int __stdcall c_ppvw_2
(
    PNTNUM    point,
    PRMNUM    param,
    float     value
);
```

## Arguments

| Argument     | Description                                                  |
|--------------|--------------------------------------------------------------|
| <b>point</b> | (in) Point type/number to be processed.                      |
| <b>param</b> | (in) Point parameter to be processed. -1 for all parameters. |
| <b>value</b> | (in) Value to be stored into the point parameter.            |

## Description

Requests a Demand scan of the specified Point and waits for the scan to complete. If after 10 seconds the scan has not replied, then a timeout will be indicated. The point is always processed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling `c_geterrno()` on page 106 will retrieve one of the following error codes:

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| [M4_INV_POINT]      | Invalid Point specified.                                            |
| [M4_INV_PARAMETER]  | Invalid Point parameter specified.                                  |
| [M4_ILLEGAL_RTU]    | There are no Controllers implemented that can process this request. |
| [M4_DEVICE_TIMEOUT] | Scan was not performed 10 second timeout interval.                  |

## See also

`c_ppv_2()` on page 226

## PritoPV()

Inserts a priority and sub-priority value into a **PARvalue** union.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/almmsg.h>

PARvalue* PritoPV(
    int2      priority,
    int2      subpriority,
    PARvalue* pvvalue
);
```

## Arguments

| Argument            | Description                                                               |
|---------------------|---------------------------------------------------------------------------|
| <b>pvvalue</b>      | (in) A pointer to a <b>PARvalue</b> structure.                            |
| <b>priority</b>     | (in) The priority value to insert into the <b>PARvalue</b> structure.     |
| <b>sub-priority</b> | (in) The sub-priority value to insert into the <b>PARvalue</b> structure. |

## Description

Inserts a priority and sub-priority value into a **PARvalue**, and then returns a pointer to the **PARvalue** passed in. This allows you to set attributes into a notification structure using calls to *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138 functions in a single line of code.

## Diagnostics

If this function is successful, the return value is a pointer back to the **PARvalue** passed in, otherwise, the return value is **NULL** and calling *c\_geterrno()* on page 106 will retrieve the following error code:

|                         |                                                               |
|-------------------------|---------------------------------------------------------------|
| <b>BUFFER_TOO_SMALL</b> | The pointer to the <b>PARvalue</b> is invalid, that is, null. |
|-------------------------|---------------------------------------------------------------|

---

## Example

See the examples in *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138.

---

## See also

*DbletoPV()* on page 98

*hsc\_insert\_attrib()* on page 131

*hsc\_insert\_attrib\_byindex()* on page 138

*Int2toPV()* on page 206

*Int4toPV()* on page 208

*RealtoPV()* on page 231

*StrtoPV()* on page 240

*TimetoPV()* on page 242

## c\_prsend\_...()

Queue file to print system for printing.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>
```

```
// Select one of the following synopses as appropriate to
// the destination

int __stdcall c_prsend_crt(
    int    crt,
    char*  filename
);

int __stdcall c_prsend_printer(
    int    printer,
    char*  filename
);
```

## Arguments

| Argument        | Description                                                                                                       |
|-----------------|-------------------------------------------------------------------------------------------------------------------|
| <b>crt</b>      | (in) CRT number.                                                                                                  |
| <b>printer</b>  | (in) printer number.                                                                                              |
| <b>filename</b> | (in) pointer to null-terminated string containing the pathname (maximum 60 characters) of the file to be printed. |

## Description

Requests the print system to print the specified file.

`c_prsend_crt`

will send the file to the demand report printer that is associated with the specified CRT.

`c_prsend_printer`

will send the file to the specified printer.

## Diagnostics

Upon successful completion, a value of **0** is returned. Otherwise, **-1** is returned and calling `c_geterrno()` on page 106 will retrieve the following error code:

|                    |                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------|
| <b>[M4_QEMPTY]</b> | The file could not be queued to the printer because the printer queue had no free records. |
|--------------------|--------------------------------------------------------------------------------------------|

---

## Example

```

#define PRINTER_NO 1
#define SAMPLE_FILENAME '...\user\sample.dat'
static char *programe='sample.c';

/* send the sample file to the printer */
if (c_prsend_printer(PRINTER_NO, SAMPLE_FILENAME) == -1)
{
    c_logmsg(programe, '123', 'c_prsend_printer
    error 0x%x', c_geterrno());
    exit(ierr);
}

```

---

## RealtPV()

Inserts a real value into a **PARvalue** union.

### C/C++ synopsis

```

#include <src/defs.h>
#include <src/almmsg.h>

PARvalue* RealtPV(
    float      real_val,
    PARvalue*  pvvalue
);

```

### Arguments

| Argument        | Description                                                       |
|-----------------|-------------------------------------------------------------------|
| <b>pvvalue</b>  | (in) A pointer to a <b>PARvalue</b> structure.                    |
| <b>real_val</b> | (in) The real value to insert into the <b>PARvalue</b> structure. |

### Description

Inserts a real value into a **PARvalue**, and then returns a pointer to the **PARvalue** passed in. This function allows you to set attributes into a notification structure using calls to *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138 functions in a single line of code.

## Diagnostics

If this function is successful, the return value is a pointer back to the **PARvalue** passed in, otherwise, the return value is **NULL** and calling *c\_geterrno()* on page 106 will retrieve the following error code:

|                         |                                                               |
|-------------------------|---------------------------------------------------------------|
| <b>BUFFER_TOO_SMALL</b> | The pointer to the <b>PARvalue</b> is invalid, that is, null. |
|-------------------------|---------------------------------------------------------------|

---

### Example

See the examples in *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138.

---

### See also

*DbletoPV()* on page 98

*hsc\_insert\_attrib()* on page 131

*hsc\_insert\_attrib\_byindex()* on page 138

*Int2toPV()* on page 206

*Int4toPV()* on page 208

*PritoPV()* on page 228

*StrtoPV()* on page 240

*TimetoPV()* on page 242

### **c\_rqtskb...()**

Requests a task if not busy.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/sn90_err.h>
#include <src/trbtbl_def>

int2 c_rqtskb(
    int    lrn
);
```

```
int2 c_rqtskb_prm(
    int   lrn,
    int2* prmblk
);
```

## Arguments

| Argument      | Description                                               |
|---------------|-----------------------------------------------------------|
| <b>lrn</b>    | (in) Logical resource number of the task to be requested. |
| <b>prmblk</b> | (in) pointer to parameter block to be passed to the task. |

## Description

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>c_<br/>rqtskb</b>          | requests the specified queued task without any parameters. If the task is already running, no action is taken.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>c_<br/>rqtskb_<br/>prm</b> | <p>requests the specified queued task and passes a parameter block. If the task is not executing, or the parameter block is zero, then the request is passed to the task. If the task is already executing and the parameter block is not zero, then the request is queued. If a request is already queued, the task is considered busy and an error code is returned.</p> <p>Most of the system's tasks take in parameters in the form of the prm structure. Note that the prm structure is defined in the file trtbl_def in the src folder. It is recommended that developers make use of this parameter block structure. To do so, first instantiate and initialize the structure with relevant data, then cast a pointer to it to an int2* in order to call this function:</p> <pre>prm my_pblk; int myLrn; myLrn = 111; //user application LRN ... return_val = c_rqtskb_prm(myLrn, (int2*) &amp;my_pblk);</pre> <p>Note that the some LRNs listed in the <b>def/src/lrns</b> file (for example, the Keyboard Service program (LRN 1) and Server Display program (LRN 21)), actually use multiple LRNs. The most important example of this is the Server Display program. Each Station is associated with its own Server Display and Keyboard Service programs. Each of these tasks use its own LRN. The Server Display programs of the first 20 Stations are assigned LRNs 21 through to 40. For example, to request the Server Display program for Station 3, you would request LRN 23.</p> <p>Stations 21-40, if assigned, use other LRNs. These can be displayed using the utility <b>usrln</b> with the options <b>-p -a</b>.</p> |

## Notes

To call up a named display in Station you need to:

1. Memset to 0 the Parameter Request Block structure.
2. Ensure the `prmbk.pathlen` is set to 0. This `pathlen` is only used for the new HMI protocol.
3. Use `c_chrint` to the full size of the path.

For example:

```
memset(&prmbk, 0, sizeof(prmbk));
prmbk.crt = Station Number (in this test case it is 1)
prmbk.param1 = 1;
prmbk.param2 = 0;
prmbk.path is set via c_chrint("testdisplay",11,prmbk.path,sizeof
(prmbk.path));
```

```
iDisplayLRN = dsply_lrn(station number);
c_rqtskb_prm(iDisplayLRN, (int2 *)&prmbk);
```

To call up a numbered display:

```
prmbk.crt = Station Number (in this test case it is 1)
prmbk.param1 = 1;
prmbk.param2 = 14;
```

```
iDisplayLRN = dsply_lrn(station number);
c_rqtskb_prm(iDisplayLRN, (int2 *)&prmbk);
```

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling `c_geterrno()` on page 106 will retrieve one of the following error codes:

|                         |                                                                       |
|-------------------------|-----------------------------------------------------------------------|
| <b>[M4_ILLEGAL_LRN]</b> | An illegal LRN has been specified or the task does not exist.         |
| <b>[M4_BUSY_TRB]</b>    | The requested tasks request block is busy, could not pass parameters. |
| <b>[INVALID_SEMVAL]</b> | The requested task has too many outstanding requests.                 |

## See also

`c_getreq()` on page 116

`c_tstskb()` on page 247

*c\_wttskb()* on page 250

## **c\_sps\_2()**

Scans a point special (without waiting for completion).

### **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/M4_err.h>
```

```
int __stdcall c_sps_2
(
    PNTNUM    point,
    PRMNUM    param
);
```

### **Arguments**

| <b>Argument</b> | <b>Description</b>                                           |
|-----------------|--------------------------------------------------------------|
| <b>point</b>    | (in) Point type/number to be processed.                      |
| <b>param</b>    | (in) Point parameter to be processed. -1 for all parameters. |

### **Description**

SPS is used to request a Demand scan of the specified Point.

The point is only processed if the scanned value has changed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

### **Diagnostics**

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                            |                                                             |
|----------------------------|-------------------------------------------------------------|
| <b>[M4_INV_POINT]</b>      | Invalid Point specified.                                    |
| <b>[M4_INV_PARAMETER]</b>  | Invalid Point parameter specified.                          |
| <b>[M4_DEVICE_TIMEOUT]</b> | Scan was not performed before a 10 second timeout interval. |

## See also

*c\_spsw\_2()* below

## c\_spsw\_2()

Scans a point special and waits for completion.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int __stdcall c_spsw_2
(
    PNTNUM    point,
    PRMNUM    param
);
```

## Arguments

| Argument     | Description                                                  |
|--------------|--------------------------------------------------------------|
| <b>point</b> | (in) Point type/number to be processed.                      |
| <b>param</b> | (in) Point parameter to be processed. -1 for all parameters. |

## Description

Requests a Demand scan of the specified Point and wait for the Scan to complete. If after 10 seconds the Scan has not replied, then a timeout will be indicated. The point is only processed if the scanned value has changed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                     |                                                    |
|---------------------|----------------------------------------------------|
| [M4_INV_POINT]      | Invalid Point specified.                           |
| [M4_INV_PARAMETER]  | Invalid Point parameter specified.                 |
| [M4_DEVICE_TIMEOUT] | Scan was not performed 10 second timeout interval. |

## See also

*c\_sps\_2()* on page 235

## c\_spv\_2()

Scans a point value (without waiting for completion).

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int __stdcall c_spv_2
(
    PNTNUM    point,
    PRMNUM    param,
    float     value
);
```

## Arguments

| Argument     | Description                                             |
|--------------|---------------------------------------------------------|
| <b>point</b> | (in) point type and/or number to be processed           |
| <b>param</b> | (in) point parameter to be processed. PV=0 through A4=7 |
| <b>value</b> | (in) value to be stored into the point parameter        |

## Description

Passes the value to the point processor for storage into the point parameter.

The point is processed only if the scanned value has changed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion, a value of **0** is returned. Otherwise, one of the following error codes is returned:

|                       |                          |
|-----------------------|--------------------------|
| <b>[M4_INV_POINT]</b> | Invalid point specified. |
|-----------------------|--------------------------|

|                            |                                                                     |
|----------------------------|---------------------------------------------------------------------|
| <b>[M4_INV_PARAMETER]</b>  | Invalid point parameter specified.                                  |
| <b>[M4_ILLEGAL_RTU]</b>    | There are no controllers implemented that can process this request. |
| <b>[M4_DEVICE_TIMEOUT]</b> | Scan was not performed before a 10 second timeout interval.         |

## See also

*c\_spvw\_2()* below

## **c\_spvw\_2()**

Scans a point value and waits for completion.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int __stdcall c_spvw_2
(
    PNTNUM    point,
    PRMNUM    param,
    float     value
);
```

## Arguments

| Argument     | Description                                              |
|--------------|----------------------------------------------------------|
| <b>point</b> | (in) point type and/or number to be processed.           |
| <b>param</b> | (in) point parameter to be processed. PV=0 through A4=7. |
| <b>value</b> | (in) value to be stored into the point parameter.        |

## Description

Passes the value to the point processor for storage into the point parameter.

The point is processed only if the scanned value has changed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion, a value of **0** is returned. Otherwise, one of the following error codes is returned:

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| [M4_INV_POINT]      | Invalid point specified.                                            |
| [M4_INV_PARAMETER]  | Invalid point parameter specified.                                  |
| [M4_ILLEGAL_RTU]    | There are no controllers implemented that can process this request. |
| [M4_DEVICE_TIMEOUT] | Scan was not performed before a 10 second timeout interval.         |

## See also

*c\_spv\_2()* on page 237

## c\_stcupd()

Updates Controller's sample time counter.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int __stdcall c_stcupd(
    int    rtu,
    int    seconds
);
```

## Arguments

| Argument       | Description                              |
|----------------|------------------------------------------|
| <b>rtu</b>     | (in) the controller number to be updated |
| <b>seconds</b> | (in) the number of seconds               |

## Description

Sets the sample time counter of a Controller. The scan task counts down this counter, and if it reaches zero the controller will be failed. A time of greater than 60 seconds must be used if automatic recovery via the diagnostic scan is required.

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling **c\_geterrno()** will retrieve one of the following error codes:

|                  |                                     |
|------------------|-------------------------------------|
| [M4_ILLEGAL_RTU] | The Controller number is not legal. |
| [M4_ILLEGAL_CHN] | The channel number is not legal.    |

## stn\_num()

Finds out the Station number of a display task given the task's LRN.

## C/C++ synopsis

```
#include <src/defs.h>
```

```
int2 stn_num(
    int2* pLrn    // (in) A pointer to the LRN
);
```

## Arguments

| Argument | Description             |
|----------|-------------------------|
| pLrn     | (in) pointer to the LRN |

## Description

Quickly determines the Station number of a particular Station's display task given its LRN.

## Diagnostics

Returns the Station number (>0) if successful. Otherwise it returns **-1**.

## See also

*dsply\_lrn()* on page 100

## StrtoPV()

Inserts a character string into a **PARvalue** union.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/almmsg.h>
#include <src/M4_err.h>

PARvalue* StrtoPV(
    const char*   string_val,
    PARvalue*    pvvalue
);
```

## Arguments

| Argument          | Description                                                            |
|-------------------|------------------------------------------------------------------------|
| <b>pvvalue</b>    | (in) A pointer to a <b>PARvalue</b> structure                          |
| <b>string_val</b> | (in) The character string to insert into the <b>PARvalue</b> structure |

## Description

Inserts a character string into a **PARvalue**, and then returns a pointer to the **PARvalue** passed in. This function allows you to set attributes into a notification structure using calls to *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138 functions in a single line of code.

## Diagnostics

If this function is successful, the return value is a pointer back to the **PARvalue** passed in, otherwise, the return value is **NULL** and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                         |                                                               |
|-------------------------|---------------------------------------------------------------|
| <b>BUFFER_TOO_SMALL</b> | The pointer to the <b>PARvalue</b> is invalid, that is, null. |
|-------------------------|---------------------------------------------------------------|

## Example

See the examples in *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138.

## See also

*DbletoPV()* on page 98

*hsc\_insert\_attrib()* on page 131

*hsc\_insert\_attrib\_byindex()* on page 138

*Int2toPV()* on page 206

*Int4toPV()* on page 208

*PritoPV()* on page 228

*RealtoPV()* on page 231

*TimetoPV()* below

## TimetoPV()

Inserts a time value into a **PARvalue** union.

### C/C++ synopsis

```
#include <src/defs.h>
#include <src/almmsg.h>
#include <src/M4_err.h>

PARvalue* TimetoPV(
    HSCTIME    time_val,
    PARvalue*  pvvalue
);
```

### Arguments

| Argument        | Description                                                      |
|-----------------|------------------------------------------------------------------|
| <b>pvvalue</b>  | (in) A pointer to a <b>PARvalue</b> structure                    |
| <b>time_val</b> | (in) The time value to insert into the <b>PARvalue</b> structure |

### Description

Inserts a time value into a **PARvalue**, and then returns a pointer to the **PARvalue** passed in. This function allows you to set attributes into a notification structure using calls to *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138 functions in a single line of code.

### Diagnostics

If this function is successful, the return value is a pointer back to the **PARvalue** passed in, otherwise, the return value is **NULL** and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

**BUFFER\_TOO\_SMALL**The pointer to the **PARvalue** is invalid, that is, null.**Example**

See the examples in *hsc\_insert\_attrib()* on page 131 and *hsc\_insert\_attrib\_byindex()* on page 138.

**See also**

*DbletoPV()* on page 98

*hsc\_insert\_attrib()* on page 131

*hsc\_insert\_attrib\_byindex()* on page 138

*Int2toPV()* on page 206

*Int4toPV()* on page 208

*PritoPV()* on page 228

*RealtoPV()* on page 231

*StrtoPV()* on page 240

**c\_tmstop()**

Stops a timer for the calling task.

**C/C++ synopsis**

```
#include <src/defs.h>

void __stdcall c_tmstop(
    int    tmridx
);
```

**Arguments**

| Argument      | Description                        |
|---------------|------------------------------------|
| <b>tmridx</b> | (in) index of which timer to stop. |

## Description

Stops the timer specified by the argument **tmridx**. This index must correspond to the return value of *c\_tmstrt\_...()* below.

## See also

*c\_tmstrt\_...()* below

*c\_trmtsk()* on page 246

## **c\_tmstrt\_...()**

Starts a timer for the calling task.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

void __stdcall c_tmstrt_single(
    int    cycle,
    int    param1,
    int    param2
);

int __stdcall c_tmstrt_cycle(
    int    cycle,
    int    param1,
    int    param2
);
```

## Arguments

| Argument      | Description                                       |
|---------------|---------------------------------------------------|
| <b>cycle</b>  | (in) time interval between executions in seconds. |
| <b>param1</b> | (in) parameter passed to task as parameter 1.     |
| <b>param2</b> | (in) parameter passed to task as parameter 2.     |

## Description

Starts a timer to request the calling task every **cycle** seconds. This is equivalent to calling *c\_rqtskb...()* on page 232 every interval. A timer is stopped by calling *c\_tmstop()* on page 243.

The arguments **param1** and **param2** are passed as words two and three of the ten word parameter block, to the task each interval. These parameters can be accessed by calling *c\_getreq()* on page 116.

|                        |                                                       |
|------------------------|-------------------------------------------------------|
| <b>c_tmstrt_single</b> | requests the specified task only once.                |
| <b>c_tmstrt_cycle</b>  | requests the specified task continuously every cycle. |

## Diagnostics

Upon successful completion the timer index is returned. Otherwise, **-1** is returned and calling *c\_geterrno()* will retrieve the following error code:

|             |                         |
|-------------|-------------------------|
| [M4_QEMPTY] | Too many timers active. |
|-------------|-------------------------|

## See also

*c\_tmstop()* on page 243

*c\_getreq()* on page 116

## c\_trm04()

Terminate task with error status and modify restart address.

## C/C++ synopsis

```
#include <src/defs.h>

void __stdcall c_trm04(
    int2    status
);
```

## Arguments

| Argument      | Description                   |
|---------------|-------------------------------|
| <b>status</b> | (in) termination error status |

## Description

Terminates the calling task and changes the restart address to the address immediately following the call to *c\_trm04()* on the previous page. The task is not terminated if it was requested while it was active. The termination error status is posted in the task request block for any *c\_wttskb()* on page 250 calls.

If the task has been marked for deletion via a *c\_deltask()* on page 97 call then the task will be removed from the system.

## See also

*c\_rqtskb...()* on page 232

*c\_tstskb()* on the next page

*c\_wttskb()* on page 250

*c\_deltask()* on page 97

## c\_trmtsk()

Terminates a task with error status.

## C/C++ synopsis

```
#include <src/defs.h>

void __stdcall c_trmtsk(
    int2    status
);
```

## Arguments

| Argument      | Description                   |
|---------------|-------------------------------|
| <b>status</b> | (in) termination error status |

## Description

Terminates the calling task and changes the restart address to the program start address. The termination error status is posted in the task request block for any *c\_wttskb()* on page 250 calls.

If the task has been marked for deletion via a *c\_deltask()* on page 97 call then the task will be removed from the system.

## See also

*c\_trm04()* on page 245

*c\_deltsk()* on page 97

## c\_tstskb()

Tests a task's status.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

void __stdcall c_tstskb(
    int    lrn
);
```

## Arguments

| Argument   | Description                                            |
|------------|--------------------------------------------------------|
| <b>lrn</b> | (in) logical resource number of the task to be tested. |

## Description

Tests the completion status of a specified task.

## Diagnostics

Upon successful completion a value of **0** is returned indicating that the specified task is dormant. Otherwise, **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                         |                                    |
|-------------------------|------------------------------------|
| <b>[M4_ILLEGAL_LRN]</b> | An illegal LRN has been specified. |
| <b>[M4_BUSY_TRB]</b>    | The specified task is active.      |

---

## Example

```
#include 'src/lrns' /* for user tasks LRN */
...

```

---

---

```

...
...
/* test to see if the first user task is dormant */
if (c_tstskb(USR1LRN) == 0)
    c_logmsg(progname, '123', 'user
    task 1 is dormant');

```

---

**See also***c\_wttskb()* on page 250*c\_rqtskb...()* on page 232*c\_trm04()* on page 245*c\_trmtsk()* on page 246**c\_upper()**

Converts a character string to upper case.

**C/C++ synopsis**

```

#include <src/defs.h>

void __stdcall c_upper(
    char*   chrstr
);

```

**Arguments**

| Argument      | Description                                           |
|---------------|-------------------------------------------------------|
| <b>string</b> | (in/out) pointer to null-terminated character string. |

**Description**

Converts a character string to upper case (7 bit ASCII). Control characters are converted to a '.' character.

**See also***c\_chrint()* on page 85*c\_intchr()* on page 209

## **c\_wdon()**

Pulses the watchdog timer for a task.

### **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/M4_err.h>

void __stdcall c_wdon(
    int    wdtidx
);
```

### **Arguments**

| Argument      | Description                                      |
|---------------|--------------------------------------------------|
| <b>wdtidx</b> | (in) index of the watchdog timer entry to pulse. |

### **Description**

Prevents the watchdog timer from timing-out the calling task.

Should only be called with a valid watchdog timer index returned from *c\_wdstrt()* below.

For more information regarding watchdog timers, see 'Modifying the activity of a task.'

### **Diagnostics**

Does not return a value and no diagnostic errors are returned if **wdtidx** is invalid.

### **See also**

*c\_wdstrt()* below

## **c\_wdstrt()**

Starts a watchdog timer for the calling task.

### **C/C++ synopsis**

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/wdstrt.h>

int __stdcall c_wdstrt(
```

```

        int     timeout,
        int     mode
    );

```

## Arguments

| Argument       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>timeout</b> | (in) time interval before watchdog timer takes action indicated by mode.                                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>mode</b>    | (in) mode of action to be taken by the watchdog timer: <ul style="list-style-type: none"> <li>■ <b>WDT_MONITOR</b> monitor timer entry only.</li> <li>■ <b>WDT_ALARM_ONCE</b> generate an alarm on first failure only.</li> <li>■ <b>WDT_ALARM</b> generate an alarm on each failure.</li> <li>■ <b>WDT_RESTART_TASK</b> restart task on first failure, reboot the server system on second failure.</li> <li>■ <b>WDT_RESTART_SYS</b> restart the server system on failure.</li> </ul> |

## Description

Enables a watchdog timer for the calling task.

Calling this routine allocates an entry in the **WDTTBL**. Each second, **WDT** decrements the timer entry. If the timer becomes zero then the action defined by the mode will be taken.

To prevent the timeout occurring, the calling task should periodically call *c\_wdon()* on the previous page to pulse reset the timer.

For more information, see 'Modifying the activity of a task.'

## Diagnostics

Upon successful completion the watchdog timer index is returned. If *c\_wdstrt()* on the previous page is unable to create a timer, it returns a **0**.

## See also

*c\_wdon()* on the previous page

*c\_wdstrt()* on the previous page

## **c\_wttskb()**

Wait for a task to become dormant.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int2 __stdcall c_wttskb(
    int    lrn
);
```

## Arguments

| Argument   | Description                                               |
|------------|-----------------------------------------------------------|
| <b>lrn</b> | (in) Logical resource number of the task to be waited on. |

## Description

Waits for the specified task to complete processing.

## Diagnostics

Upon successful completion the specified tasks termination error status is returned. Otherwise, the following error code is returned:

|                         |                                    |
|-------------------------|------------------------------------|
| <b>[M4_ILLEGAL_LRN]</b> | An illegal LRN has been specified. |
|-------------------------|------------------------------------|

## See also

*c\_tstskb()* on page 247

*c\_rqtskb...()* on page 232

*c\_trm04()* on page 245

*c\_trmtask()* on page 246

## Backward-compatible functions

The following functions are available for backwards compatibility.

*c\_badpar()* on the next page

*c\_getstpar\_...()* on page 253

*c\_pps()* on page 257

*c\_ppsw()* on page 258

*c\_ppv()* on page 260

*c\_ppvw()* on page 261

*c\_sps()* on page 263

*c\_spsw()* on page 264

*c\_spv()* on page 265

*c\_spvw()* on page 267

## **c\_badpar()**

---

### **Attention:**

**c\_badpar()** is deprecated and may be removed in a future release. It is provided for backward compatibility purposes only. When used with an Experion HS server release R400 or later **c\_badpar()** will only be able to access points in the range:  $1 \leq \text{point number} \leq 65,000$ . Checking the return value of **hsc\_param\_value()** provides the same functionality as **c\_badpar()**.

---

Tests for a bad parameter value.

### **C/C++ synopsis**

```
#include <src/defs.h>

int __stdcall c_badpar
(
    PNTNUM16 point,
    PRMNUM   param
);
```

### **Arguments**

| <b>Argument</b> | <b>Description</b>                   |
|-----------------|--------------------------------------|
| <b>point</b>    | (in) point type/number to be tested. |
| <b>param</b>    | (in) parameter to be tested.         |

### **Description**

Returns **TRUE** if the system is not running, if the specified point is not implemented, or if the parameter value is in error; otherwise **FALSE** is returned.

**See also***c\_mzero()* on page 218*hsc\_param\_value()* on page 167**c\_gethstpar\_...()****Attention:**


---

**c\_gethstpar()** is deprecated and may be removed in a future release. It is provided for backward compatibility purposes only. When used with an Experion HS server release R400 or later **c\_gethstpar()** will only be able to access points in the range:  $1 \leq \text{point number} \leq 65,000$ . The replacement function **c\_gethstpar\_2()** should be used instead.

---

Gets the history interface parameters.

**C/C++ synopsis**

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/gethst.h>

// Select one of the following synopses as appropriate
// to
// the type of request being sent

int __stdcall c_gethstpar_date
(
    int    type,
    int    date,
    float  time,
    int    numhst,
    uint2* points,
    uint2* params,
    int    numpnt,
    char*  archive,
    float* values
);
```

```
int __stdcall c_gethstpar_ofst
(
    int     type,
    int     offset,
    int     numhst,
    uint2*  points,
    uint2*  params,
    int     numpnt,
    char*   archive,
    float*  values
);
```

### Arguments

| Argument       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>type</b>    | (in) history type (see Description).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>date</b>    | (in) start date of history to retrieve in Julian days (number of days since 1 Jan 1981).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <b>time</b>    | (in) start time of history to retrieve in seconds since midnight.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>offset</b>  | (in) offset from latest history value in history intervals (where offset=1 is the most recent history value).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <b>numhst</b>  | (in) number of history values to be returned per Point.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>points</b>  | (in) array of Point type/numbers to process (maximum of 100 elements).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>params</b>  | (in) array of point parameters to process. Each parameter is associated with the corresponding entry in the points array. The possible parameters are defined in the file 'parameters' in the <b>def</b> folder (maximum 100 elements).                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>numpnt</b>  | (in) number of Points to be processed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>archive</b> | (in) pointer to a null-terminated string containing the folder name of the archive files relative to the archive folder. A NULL pointer implies that the system will use current history and any archive files that correspond to the value of the date and time parameters. The archive files are found in <b>&lt;data folder&gt;\Honeywell\Experion HS\Server\data\archive</b> . Where <b>&lt;data folder&gt;</b> is the location where Experion data is stored. For default installations, this is <b>C:\ProgramData</b> .<br><br>For example, to access the files in <b>&lt;data folder&gt;\Honeywell\Experion HS\Server\archive\ay2012m09d26h11r008</b> , the archive argument is <b>ay2012m09d26h11r008</b> . |
| <b>values</b>  | (out) two dimensional array large enough to accept history values. If there is no history for the requested time or if the data was bad, then -0.0 is stored in the array. Sized <b>numpnt * numhst</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## Description

Used to retrieve a particular type of history values for specified Points and time in history. History will be retrieved from a specified time or Offset going backwards in time **numhst** intervals for each Point specified.

|                         |                                                                                    |
|-------------------------|------------------------------------------------------------------------------------|
| <b>c_gethstpar_date</b> | retrieves history values from a specified date and time.                           |
| <b>c_gethstpar_ofst</b> | retrieves history values from a specified number of history intervals in the past. |

The history values are stored in sequence in the **values** array. **values[x][y]** represents the **yth** history value for the **xth** point.

The history type is specified by using one of the following values:

| Value              | Description                       |
|--------------------|-----------------------------------|
| <b>HST_1MIN</b>    | one minute standard history       |
| <b>HST_6MIN</b>    | six minute standard history       |
| <b>HST_1HOUR</b>   | one hour standard history         |
| <b>HST_8HOUR</b>   | eight hour standard history       |
| <b>HST_24HOUR</b>  | twenty four hour standard history |
| <b>HST_5SECF</b>   | Fast history                      |
| <b>HST_1HOURE</b>  | one hour extended history         |
| <b>HST_8HOURE</b>  | eight hour extended history       |
| <b>HST_24HOURE</b> | twenty four hour extended history |

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned, and calling **c\_geterrno()** will return one of the following error codes:

|                         |                                                       |
|-------------------------|-------------------------------------------------------|
| <b>[M4_ILLEGAL_VAL]</b> | Illegal number of Points or history values specified. |
| <b>[M4_ILLEGAL_HST]</b> | Illegal history type or interval specified.           |
| <b>[M4_VAL_NOT_FND]</b> | value not found in history.                           |

---

**Example**

```
#include <src/defs.h>
#include <src/M4_err.h>
#include <src/gethst.h>
#include 'parameters'
#define NHST 50
#define NPNT 3

int errcode; /* error code */
int date; /* julian date */
float time; /* seconds from midnight */
int year; /* year from OAD */
int month; /* month (1 - 12) */
int day; /* day (1 - 31) */
int i; /* iteration */
int hour; /* hour (0 - 23) */
int minute; /* min (0 - 59) */
uint2 points[NPNT]; /* point
numbers */
uint2 params[NPNT]; /* parameters */
float values[NPNT][NHST]; /*
history values */
. . .
. . .
. . .
/* attach database */
if (c_gbload())
{
    errcode = c_geterrno();
    c_logmsg(progname, '123', 'c_gbload error %#x', errcode);
    exit(errcode);
}

/*get the point numbers of the following points*/
c_getpnt('C1TEMP', &points[0]);
c_getpnt('C1PRES', &points[1]);
c_getpnt('C2TIME', &points[2]);
```

---

---

```

/*set up for all PV parameters*/
for (i=0; i<NPNT; i++)
    params[i]=PV;

/*set up seconds since midnight and julian date*/
time = (hour* 60+minute)* 60;
date = c_gtoj(year, month, day);
. . .
. . .
. . .
/*retrieve the history*/
if (c_gethstpar_date(type, date, time, nhst, params, points, npnt,
NULL, values) == -1)
{
errcode = c_geterrno();
c_logmsg(progname, '123', ' c_gethstpar_date error %#x',errcode);exit
(errcode);
}
. . .
. . .
. . .

```

---

**See also**

*hsc\_param\_values()* on page 170

**c\_pps()****Attention:**

**c\_pps()** is deprecated and may be removed in a future release. It is provided for backward compatibility purposes only. When used with an Experion HS server release R400 or later **c\_pps()** will only be able to access points in the range: 1<= point number <= 65,000. The replacement function **c\_pps\_2()** should be used instead.

---

Processes a point special (without waiting for completion).

**C/C++ synopsis**

```

#include <src/defs.h>
#include <scr/M4_err.h>

int __stdcall c_pps
(
    PNTNUM16 point,
    int      param,
    int*     status
);

```

## Arguments

| Argument      | Description                                                  |
|---------------|--------------------------------------------------------------|
| <b>point</b>  | (in) Point type/number to be processed.                      |
| <b>param</b>  | (in) Point parameter to be processed. -1 for all parameters. |
| <b>status</b> | (out) return error code.                                     |

## Description

Requests a demand scan of the specified point.

The point is always processed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion, a value of **0** is returned in status. Otherwise, one of the following error codes is returned:

|                     |                                                            |
|---------------------|------------------------------------------------------------|
| [M4_INV_POINT]      | Invalid point specified                                    |
| [MR_INV_PARAMETER]  | Invalid point parameter specified                          |
| [M4_DEVICE_TIMEOUT] | Scan was not performed before a 10 second timeout interval |

## See also

*c\_ppsw()* below

*c\_pps\_2()* on page 223

## **c\_ppsw()**

**Attention:**

**c\_ppsw()** is deprecated and may be removed in a future release. It is provided for backward compatibility purposes only. When used with an Experion HS server release R400 or later **c\_ppsw()** will only be able to access points in the range: 1<= point number <= 65,000. The replacement function **c\_ppsw\_2()** should be used instead.

Processes a point special and waits for completion.

**C/C++ synopsis**

```
#include <src/defs.h>
#include <scr/M4_err.h>

int __stdcall c_ppsw
(
    PNTNUM16 point,
    int      param,
    int*     status
);
```

**Arguments**

| Argument      | Description                                                  |
|---------------|--------------------------------------------------------------|
| <b>point</b>  | (in) point type/number to be processed.                      |
| <b>param</b>  | (in) point parameter to be processed. -1 for all parameters. |
| <b>status</b> | (out) return error code.                                     |

**Description**

Requests a demand scan of the specified point and wait for the scan to complete. If, after 10 seconds, the scan has not replied, a timeout will be indicated.

The point is always processed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion, a value of **0** is returned in status. Otherwise, one of the following error codes is returned:

|                     |                                                             |
|---------------------|-------------------------------------------------------------|
| [M4_INV_POINT}      | Invalid point specified.                                    |
| [M4_INV_PARAMETER]  | Invalid point parameter specified.                          |
| [M4_DEVICE_TIMEOUT] | Scan was not performed before a 10 second timeout interval. |

## See also

*c\_pps()* on page 257

*c\_ppsw\_2()* on page 225

## c\_ppv()

---

### Attention:

**c\_ppv()** is deprecated and may be removed in a future release. It is provided for backward compatibility purposes only. When used with an Experion HS server release R400 or later **c\_ppv()** will only be able to access points in the range: 1<= point number <= 65,000. The replacement function **c\_ppv\_2()** should be used instead.

---

Processes a point value (without waiting for completion).

## C/C++ synopsis

```
#include <src/defs.h>
#include <scr/M4_err.h>

int __stdcall c_ppv
(
    PNTNUM16 point,
    int      param,
    float    value
);
```

## Arguments

| Argument     | Description                                                  |
|--------------|--------------------------------------------------------------|
| <b>point</b> | (in) Point type/number to be processed.                      |
| <b>param</b> | (in) Point parameter to be processed. -1 for all parameters. |
| <b>value</b> | (in) Value to be stored into the point parameter.            |

## Description

Requests a Demand scan of the specified Point.

The point is always processed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| [M4_INV_POINT]      | Invalid Point specified.                                            |
| [M4_INV_PARAMETER]  | Invalid Point parameter specified.                                  |
| [M4_ILLEGAL_RTU]    | There are no Controllers implemented that can process this request. |
| [M4_DEVICE_TIMEOUT] | Scan was not performed before a 10 second timeout interval.         |

## See also

*c\_ppvw()* below

*c\_ppv\_2()* on page 226

## **c\_ppvw()**

---

**Attention:**

---

---

**c\_ppvw()** is deprecated and may be removed in a future release. It is provided for backward compatibility purposes only. When used with an Experion HS server release R400 or later **c\_ppvw()** will only be able to access points in the range:  $1 \leq \text{point number} \leq 65,000$ . The replacement function **c\_ppvw\_2()** should be used instead.

---

Processes a point value and waits for completion.

## C/C++ synopsis

```
#include <src/defs.h>
#include <scr/M4_err.h>

int __stdcall c_ppvw
(
    PNTNUM16 point,
    int      param,
    float    value
);
```

## Arguments

| Argument     | Description                                                  |
|--------------|--------------------------------------------------------------|
| <b>point</b> | (in) Point type/number to be processed.                      |
| <b>param</b> | (in) Point parameter to be processed. -1 for all parameters. |
| <b>value</b> | (in) Value to be stored into the point parameter.            |

## Description

Requests a Demand scan of the specified Point and waits for the scan to complete. If after 10 seconds the scan has not replied, then a timeout will be indicated. The point is always processed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                            |                                                                     |
|----------------------------|---------------------------------------------------------------------|
| <b>[M4_INV_POINT]</b>      | Invalid Point specified.                                            |
| <b>[M4_INV_PARAMETER]</b>  | Invalid Point parameter specified.                                  |
| <b>[M4_ILLEGAL_RTU]</b>    | There are no Controllers implemented that can process this request. |
| <b>[M4_DEVICE_TIMEOUT]</b> | Scan was not performed 10 second timeout interval.                  |

## See also

*c\_ppv()* on page 260

*c\_ppvw\_2()* on page 227

## **c\_sps()**

---

### Attention:

**c\_sps()** is deprecated and may be removed in a future release. It is provided for backward compatibility purposes only. When used with an Experion HS server release R400 or later **c\_sps()** will only be able to access points in the range: 1 <= point number <= 65,000. The replacement function **c\_sps\_2()** should be used instead.

---

Scans a point special (without waiting for completion).

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int __stdcall c_sps
(
    PNTNUM16  point,
    int       param
);
```

## Arguments

| Argument     | Description                                                  |
|--------------|--------------------------------------------------------------|
| <b>point</b> | (in) Point type/number to be processed.                      |
| <b>param</b> | (in) Point parameter to be processed. -1 for all parameters. |

## Description

SPS is used to request a Demand scan of the specified Point.

The point is only processed if the scanned value has changed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                     |                                                             |
|---------------------|-------------------------------------------------------------|
| [M4_INV_POINT]      | Invalid Point specified.                                    |
| [M4_INV_PARAMETER]  | Invalid Point parameter specified.                          |
| [M4_DEVICE_TIMEOUT] | Scan was not performed before a 10 second timeout interval. |

## See also

*c\_spsw()* below

*c\_sps\_2()* on page 235

## **c\_spsw()**

---

### Attention:

**c\_spsw()** is deprecated and may be removed in a future release. It is provided for backward compatibility purposes only. When used with an Experion HS server release R400 or later **c\_spsw()** will only be able to access points in the range: 1 <= point number <= 65,000. The replacement function **c\_spsw\_2()** should be used instead.

---

Scans a point special and waits for completion.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int __stdcall c_spsw
(
    PNTNUM16 point,
    int      param
);
```

## Arguments

| Argument     | Description                                                  |
|--------------|--------------------------------------------------------------|
| <b>point</b> | (in) Point type/number to be processed.                      |
| <b>param</b> | (in) Point parameter to be processed. -1 for all parameters. |

## Description

Requests a Demand scan of the specified Point and wait for the Scan to complete. If after 10 seconds the Scan has not replied, then a timeout will be indicated. The point is only processed if the scanned value has changed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion a value of **0** is returned. Otherwise, **-1** is returned and calling *c\_geterrno()* on page 106 will retrieve one of the following error codes:

|                     |                                                    |
|---------------------|----------------------------------------------------|
| [M4_INV_POINT]      | Invalid Point specified.                           |
| [M4_INV_PARAMETER]  | Invalid Point parameter specified.                 |
| [M4_DEVICE_TIMEOUT] | Scan was not performed 10 second timeout interval. |

## See also

*c\_sps()* on page 263

*c\_spsw\_2()* on page 236

## c\_spv()

**Attention:**

**c\_spv()** is deprecated and may be removed in a future release. It is provided for backward compatibility purposes only. When used with an Experion HS server release R400 or later **c\_spv()** will only be able to access points in the range: 1<= point number <= 65,000. The replacement function **c\_spv\_2()** should be used instead.

Scans a point value (without waiting for completion).

**C/C++ synopsis**

```
#include <src/defs.h>
#include <src/M4_err.h>

int __stdcall c_spv
(
    PNTNUM16 point,
    int      param,
    float    value
);
```

**Arguments**

| Argument     | Description                                             |
|--------------|---------------------------------------------------------|
| <b>point</b> | (in) point type and/or number to be processed           |
| <b>param</b> | (in) point parameter to be processed. PV=0 through A4=7 |
| <b>value</b> | (in) value to be stored into the point parameter        |

**Description**

Passes the value to the point processor for storage into the point parameter.

The point is processed only if the scanned value has changed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

**Diagnostics**

Upon successful completion, a value of **0** is returned. Otherwise, one of the following error codes is returned:

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| [M4_INV_POINT]      | Invalid point specified.                                            |
| [M4_INV_PARAMETER]  | Invalid point parameter specified.                                  |
| [M4_ILLEGAL_RTU]    | There are no controllers implemented that can process this request. |
| [M4_DEVICE_TIMEOUT] | Scan was not performed before a 10 second timeout interval.         |

## See also

*c\_spvw()* below

*c\_spv\_2()* on page 237

## **c\_spvw()**

---

### Attention:

**c\_spvw()** is deprecated and may be removed in a future release. It is provided for backward compatibility purposes only. When used with an Experion HS server release R400 or later **c\_spvw()** will only be able to access points in the range: 1 <= point number <= 65,000. The replacement function **c\_spvw\_2()** should be used instead.

---

Scans a point value and waits for completion.

## C/C++ synopsis

```
#include <src/defs.h>
#include <src/M4_err.h>

int __stdcall c_spvw
(
    PNTNUM16 point,
    int      param,
    float    value
);
```

## Arguments

| Argument     | Description                                              |
|--------------|----------------------------------------------------------|
| <b>point</b> | (in) point type and/or number to be processed.           |
| <b>param</b> | (in) point parameter to be processed. PV=0 through A4=7. |
| <b>value</b> | (in) value to be stored into the point parameter.        |

## Description

Passes the value to the point processor for storage into the point parameter.

The point is processed only if the scanned value has changed.

This function is not applicable to Honeywell Process Controller points, Remote points, Container points or System Interface Points (PSA).

## Diagnostics

Upon successful completion, a value of **0** is returned. Otherwise, one of the following error codes is returned:

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| [M4_INV_POINT]      | Invalid point specified.                                            |
| [M4_INV_PARAMETER]  | Invalid point parameter specified.                                  |
| [M4_ILLEGAL_RTU]    | There are no controllers implemented that can process this request. |
| [M4_DEVICE_TIMEOUT] | Scan was not performed before a 10 second timeout interval.         |

## See also

*c\_spv()* on page 265

*c\_spvw\_2()* on page 238

## Examples

There are several examples located in <data folder>\Honeywell\Experion HS\Server\user\examples\src. These can be used as a basis for your own programs.

The examples are:

| <b>Example</b> | <b>Description</b>                                                                                                                        |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <b>test1.c</b> | a very simple task which prints 'Hello World' every time it is requested.                                                                 |
| <b>test2.c</b> | a simple task that generates 3 alarms when requested.                                                                                     |
| <b>test3.c</b> | a task that demonstrates dealing with points.                                                                                             |
| <b>test4.c</b> | a simple task that demonstrates the use of user tables.                                                                                   |
| <b>test5.c</b> | a more completed task that demonstrates the use of user tables.                                                                           |
| <b>test6.c</b> | a utility that demonstrates dealing with points.                                                                                          |
| <b>test7.c</b> | a complicated task demonstrating the use of watchdog timers, scan point special, <b>hsc_param_values</b> and <b>hsc_param_value_put</b> . |
| <b>test8.c</b> | a simple task that shows the information passed on the prmbk when the task is requested.                                                  |

---

## Network API reference

This section describes how to write applications for Experion using the Network API.

### Prerequisites

Before writing network applications for Experion, you need to:

- Install Experion and third-party software as described in the *Installation Guide*.
- Review the current security settings for Network API on the Security tab of the Server Wide Settings display and ensure that **Disable writes via Network API** is enabled if you do not want data written to the server via the Network API. For more information, see the topic 'Security tab, server wide settings' in the chapter 'Customizing Stations' in the .
- Be familiar with user access and file management as described in the .

### Prerequisite skills

This guide assumes that you are an experienced programmer with a good understanding of either C, C++, or Visual Basic.

It also assumes that you are familiar with the Microsoft Windows development environment and know how to edit, compile and link applications.

---

# Network application programming

## About the Network API

The Network API has two components:

- **Network Server Option.** Enables remote computers to read and write information stored in the Experion server database. The Network Server Option runs on the server and is required for any of the network options to work (for example, Network API, Network Scan Task, and Microsoft Excel Data Exchange).

After the Network Server Option software is installed, it listens for any requests from other computers. The Network Server Option processes these requests on the behalf of the remote computer whether it be a request for a function to be performed or information to be returned from the database.

---

### Attention:

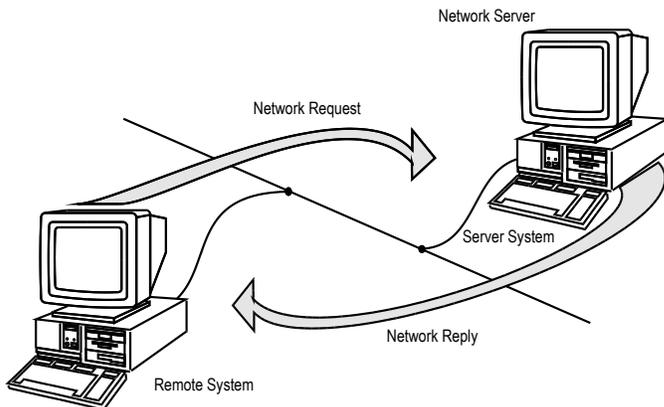
If you do not want data to be written to the server via the Network API, select **Disable writes via Network API**. For more information, see “Security tab, server wide settings” in the *Server and Client Configuration Guide*. Note that during a clean installation of Experion, writes via the Network API will be disabled by default.

---

- **Files.** Allow your program to interact with the Network Server Option. These files are comprised of C/C++ libraries, header files, VB files, Dynamic Link Libraries, documentation, and sample source programs which are all designed to help you easily create a network application.

The network applications you develop can only run on 32-bit Windows environments. Network applications act as clients to the Network Server Option and can read and write values in the Experion server database via the network.

### Network server



### Specifying a network server in a redundant system

When specifying the name of the server in your Network API application, use only the base server name for a redundant/dual-network system.

For example, you would refer only to **hsserv** when creating an application and never to a specific computer such as **hsserva**. In this way, redundancy is handled transparently whenever there is a server or network failure.

Do not use IP addresses where redundancy is required. Transparent failover cannot operate if you use IP addresses. You should only consider using IP addresses only on a single-network, single-server system.

Ensure that you correctly configure the `%systemroot%\system32\drivers\etc\Hosts` file on your client computer properly. This file provides a mapping from host names to their internet addresses.

In a single-server, single-network system you can use just the basename. The following configurations, however, require that you use more than just the basename:

| System architecture              | Host names in hosts file                                                                   |
|----------------------------------|--------------------------------------------------------------------------------------------|
| Redundant server, single network | Basename appended with a or b ( <i>hsserva, hsservb</i> )                                  |
| Single server, dual network      | Basename appended with 0 or 1 ( <i>hsserv0, hsserv1</i> )                                  |
| Redundant server, dual network   | Basename appended with a or b and 0 or 1 ( <i>servera0, servera1, serverb0, serverb1</i> ) |

On any of the redundant/dual-network configurations above do not add the basename (server) to the hosts file.

Support for redundancy with the Network API is limited to Windows.

### Summary of Network API functions

The Network API includes functions that allow you to get and put various values into the server database.

| Function                          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Generate Alarms and Events</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| rhsc_notifications                | Use to remotely generate alarms and events. The various text fields are formatted into a standard event log line on the server. The nPriority field defines the behavior on the server. An error will be returned by this function if writes via the Network API have been disabled. For more information, see <b>Disable writes via Network API</b> in “Security tab, server wide settings” in the <i>Server and Client Configuration</i> |

| Function                             | Description                                                                                                                                                                                                                                                                                                                                                   |
|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                      | <i>Guide.</i>                                                                                                                                                                                                                                                                                                                                                 |
| <b>Point Parameter Access</b>        |                                                                                                                                                                                                                                                                                                                                                               |
| rhsc_param_value_bynames             | Similar to rhsc_param_values_2 but provides a simpler calling interface but is less efficient.                                                                                                                                                                                                                                                                |
| rhsc_param_value_put_bynames         | Similar to rhsc_param_value_puts_2 but provides a simpler calling interface but is less efficient. An error will be returned by this function if writes via the Network API have been disabled. For more information, see <b>Disable writes via Network API</b> in “Security tab, server wide settings” in the <i>Server and Client Configuration Guide</i> . |
| <b>Historical Information Access</b> |                                                                                                                                                                                                                                                                                                                                                               |
| rhsc_param_hist_dates_2              | Read a block of history data from the database starting from a specified date and time.                                                                                                                                                                                                                                                                       |
| rhsc_param_hist_offsets_2            | Read a block of history data from the database starting from a specified offset in the history database.                                                                                                                                                                                                                                                      |
| rhsc_param_hist_date_bynames         | Read a block of history data from the database using point and parameter names starting from a specified date and time. Similar to rhsc_param_hist_dates_2 but provides a simpler calling interface but is less efficient.                                                                                                                                    |
| rhsc_param_hist_offset_bynames       | Read a block of history data from the database using point and parameter names starting from a specified offset in the history database. Similar to rhsc_param_hist_offsets_2 but provides a simpler calling interface but is less efficient.                                                                                                                 |
| <b>User File Information Access</b>  |                                                                                                                                                                                                                                                                                                                                                               |
| rgetdat                              | Read a list of fields from the user files of the database.                                                                                                                                                                                                                                                                                                    |
| rputdat                              | Write a list of fields to the user files of the database. An error will be returned by this function if writes via the Network API have been disabled. For more information, see <b>Disable writes via Network API</b> in “Security tab, server wide settings” in the <i>Server and Client Configuration Guide</i> .                                          |
| <b>Error String Lookup</b>           |                                                                                                                                                                                                                                                                                                                                                               |
| hsc_napierrstr2                      | Visual Basic only. Look up the error string for an error number. (Preferred command in place of hsc_napierrstr which has been retained only for backward compatibility.)                                                                                                                                                                                      |
| hsc_                                 | Look up the error string for an error number.                                                                                                                                                                                                                                                                                                                 |

| Function   | Description |
|------------|-------------|
| napierrstr |             |

## Using the Network API

Applications that use the Network API attempt to perform a single sign-on using the credentials of the calling application. This sign-on therefore needs to be configured as an *Operator* within the Experionserver, either directly or by using Windows group membership. For more information about configuring Operators, see the “Configuring system security” section in the *Server and Client Configuration Guide*.

A minimum security level of **OPER** is required to perform any writes using the Network API. If Network API writes are configured to trigger an Event, the Windows account being used to run the Network API client will also be used in the Event. With Network API writes enabled, all point writes will trigger an Event and many database writes will be listed as Events (as would occur if a database write was through Station).

| To:                             | Go to:                                                              |
|---------------------------------|---------------------------------------------------------------------|
| Determine the point numbers     | <i>Determining point numbers</i> below                              |
| Determine the parameter numbers | <i>Determining parameter numbers</i> on the next page               |
| Access point parameters         | <i>Accessing point parameters</i> on page 276                       |
| Access historical information   | <i>Accessing historical information</i> on page 277                 |
| Access user tables              | <i>Accessing user table data</i> on page 279                        |
| Look up error strings           | <i>Looking up error strings</i> on page 284                         |
| Access parameter values by name | <i>Functions for accessing parameter values by name</i> on page 285 |

### Determining point numbers

Generally, Network API functions use the point number that is used by the server in order to identify the point, rather than the point name. This internal number is stored in the server database. The point number is specific to each computer and cannot be used interchangeably on different servers.

To resolve the point name to the point number, use the function **rhsc\_point\_numbers\_2** for all the points you want to access. It is best to call this function in the initialization code of your application.

**rhsc\_point\_numbers\_2** is called with the name of the server, the number of point IDs to convert, and a data structure containing the list of point IDs. The corresponding point numbers are then returned inside this data structure by the call. Note that you should check

the function return value. If any individual request for a point number failed, the return value will be a warning that indicates a partial function fail has occurred. To find out which request failed and why, check the **fStatus** field of the data structure for each point number returned.

---

An example of using **rhsc\_point\_numbers\_2** for C is located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\C\Napiitst** project.

An example of using **rhsc\_point\_numbers\_2** for C++ is located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\C\Mfcnetapitest\Mfcnetapitest** project.

An example of using **rhsc\_point\_numbers\_2** for VB can be located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\Vb\Vbnetapitst** project.

---

## Functions that do not require a point name to number resolution

The following functions do not require you to resolve point names to point numbers before being called:

- *rhsc\_param\_value\_bynames* on page 314
- *rhsc\_param\_value\_put\_bynames* on page 318
- *rhsc\_param\_hist\_date\_bynames* on page 298
- *rhsc\_param\_hist\_offset\_bynames* on page 298

## Determining parameter numbers

Just as most Network API functions require point names to be resolved into point numbers, most parameters used by Network API functions need to have their parameter names resolved into parameter numbers. The parameter number is an internal number used in the server database to represent a parameter of a single point. Note that this parameter number is only valid for a specific point on a given server and cannot be used interchangeably between points on other servers even for identical parameter names.

To resolve a parameter name to a parameter number, use the function **rhsc\_param\_numbers\_2**. It is best to call this function in the initialization code of your application for each parameter of every point you want to access.

**rhsc\_param\_numbers\_2** is called with the name of the server, the number of parameter names to convert, and a data structure containing the list of point number and parameter name pairs. The corresponding parameter numbers are returned inside this data structure by the call.

Check the return value for any warning that a partial function fail has occurred. To find out which request failed and why, check the **fStatus** field of the data structure for each parameter number returned.

---

An example of using **rhsc\_param\_numbers\_2** for C is located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\C\Napi\st** project.

An example of using **rhsc\_param\_numbers\_2** for C++ is located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\C\Mfcnetapitest\Mfcnetapi test** project.

An example of using **rhsc\_param\_numbers\_2** for VB can be located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\Vb\Vbnetapitst** project.

---

## Functions that do not require parameter name to number resolution

The following functions do not require resolution of parameter names to parameter numbers before being called:

- *rhsc\_param\_value\_bynames* on page 314
- *rhsc\_param\_value\_put\_bynames* on page 318
- *rhsc\_param\_hist\_date\_bynames* on page 298
- *rhsc\_param\_hist\_offset\_bynames* on page 298

## Accessing point parameters

When you have resolved the point and parameter numbers for all the parameters you are interested in, values for these parameters can be retrieved using the function **rhsc\_param\_values\_2**. This function is called with the name of the server, a subscription period, the number of point parameter values to retrieve, and a data structure containing the list of point number/parameter number/offset tuples. The value of the parameter is returned in this data structure by the call, together with the type of the value. Check the return value for a partial function fail in case any of the requests for a parameter value failed.

The **rhsc\_param\_values\_2** function can acquire a list of parameter values with a mix of data types. For each parameter value requested, the parameter value data type is returned with the parameter value so that the user can determine how to handle the value. The data types supported are: DT\_CHAR, DT\_INT2, DT\_INT4, DT\_REAL, DT\_DBLE and DT\_ENUM.

**rhsc\_param\_value\_puts\_2** is a function for setting parameter values on the server. This function is called with the name of the server, the number of point parameters to write to the server, and a data structure (identical to that used by **rhsc\_param\_values\_2**) containing a list of point number/parameter number/offset/parameter value/parameter type tuples. The status of each write is returned in this data structure by the call. Check the return value for a partial function fail in case an individual write failed on the server.

Although **rhsc\_param\_value\_puts\_2** is a list based function, there is no implication that it should be used as a sequential write function. If any individual put fails, the function will not prevent the remaining writes from occurring. The function will instead continue to write values to the remaining point parameters in the list.

Both of these point access functions, **rhsc\_param\_values\_2** and **rhsc\_param\_value\_puts\_2**, require the user to be aware of memory management. The user is responsible for allocating space in the data structure used by these functions and for freeing this space before exiting the network application.

For the **rhsc\_param\_values\_2** call, the subscription period field is used to indicate the frequency at which your code will request the data. This allows the server to optimize its scanning strategies for the data you are interested in. If you are only using this routine occasionally, use the constant **NADS\_READ\_CACHE** so the server does not proceed with the optimization process.

The functions *rhsc\_param\_value\_bynames* on page 314 and *rhsc\_param\_value\_put\_bynames* on page 318 are alternative functions that perform the same tasks as **rhsc\_param\_values\_2** and **rhsc\_param\_value\_puts\_2**. There are performance costs associated with using these functions and it is preferable, where possible and when performance is a priority, to use the **rhsc\_param\_values\_2** and **rhsc\_param\_value\_puts\_2** functions instead.

---

Examples of using **rhsc\_param\_values\_2** and **rhsc\_param\_value\_puts\_2** for C are located in the **<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\C\Napitst** project.

An example of using **rhsc\_param\_values\_2** and **rhsc\_param\_value\_puts\_2** for C++ is located in the **<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\C\Mfcnetapitest\Mfcnetapitest** project.

An example of using **rhsc\_param\_values\_2** and **rhsc\_param\_value\_puts\_2** for VB can be located in the **<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\VB\VBnetapitst** project.

---

### Accessing historical information

The point and parameter numbers returned by the **rhsc\_point\_numbers\_2** and **rhsc\_param\_numbers\_2** calls can be used to identify the points for which you want to retrieve historical data as well. The functions provided to do this (in the Network API) are: **rhsc\_**

### **param\_hist\_dates\_2** and **rhsc\_param\_hist\_offsets\_2**.

The function **rhsc\_param\_hist\_offsets\_2** is used when you know the sample offset from which you want to retrieve history. It is called with the name of the server system, and a data structure containing the history type, offset, number of samples, and a list of points you want to obtain history from.

The allowable **hist\_type** values are defined in the header files **nads\_def.h** and **nif\_typ.bas** which are found in the **netapi\include** folder. Note that it is the responsibility of the calling function to allocate space for the history value structure.

The function **rhsc\_param\_hist\_dates\_2** is used when you know the date and time for which you want to retrieve history. It is similar to **rhsc\_param\_hist\_offsets\_2** but uses the date and time rather than the sample offset.

The function is called with the name of the server and a data structure containing: the history type, date, time, number of samples, and a list of points you want to obtain history from.

---

#### **Attention:**

The maximum number of points that can be processed with one call to **rhsc\_param\_hist\_x** is 20.

---

The functions **rhsc\_param\_hist\_date\_bynames** and **rhsc\_param\_hist\_offset\_bynames** are the functional equivalents of **rhsc\_param\_hist\_dates\_2** and **rhsc\_param\_hist\_offsets\_2** except that point and parameter names are used instead of point and parameter numbers. All point and parameter name resolutions are performed by the server.

There are performance costs associated with using the **rhsc\_param\_hist\_date\_bynames** and **rhsc\_param\_hist\_offset\_bynames** functions because of the extra work required of the server and the extra network traffic caused by passing names instead of numbers to the server across the network.

---

An example of using **rhsc\_param\_hist\_x\_2** for C can be located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\C\Napi\_tst** project.

An example of using **rhsc\_param\_hist\_x\_2** for C++ is located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\C\Mfcnetapitest\Mfcnetapitest** project.

An example of using **rhsc\_param\_hist\_x\_2** for VB can be located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\VB\VBnetapitst** project.

---

### Accessing user table data

A user table is a convenient way of storing application-specific data in the server database. Many of the server functions are able to read and write information from the user tables, thereby enabling you to extend the capabilities of the Server.

### Defining the user table layout

After the user table has been configured, you will need to layout the individual fields in the records. This layout information is to be used by the Network API so that it can determine how to interpret the user table.

Think of a single record as a series of individual fields lined up against one another. The server supports the following types of data fields:

| Data field                   | Description                              |
|------------------------------|------------------------------------------|
| INT2 (VB equivalent Integer) | A two byte signed integer                |
| INT4 (VB equivalent Long)    | A four byte signed integer               |
| REAL4 (VB equivalent Single) | A four byte IEEE floating point number   |
| REAL8 (VB equivalent Double) | An eight byte IEEE floating point number |

When defining the record layout of a user table, list the fields in consecutive order with their data type, description and calculate their word offset from the beginning of the record.

### Accessing one field at a time

The function **rgetdat** is used to read a series of fields from the user tables in a remote Server database. It is called with the name of the Server, the number of fields to read and an array of data structures defining the fields to retrieve. The fields listed in the array may be from any table or any record within a table.

The function **rputdat** is used to write a series of fields into the user tables in a remote Server database. This function is called with the same arguments as the **rputdat** function.

---

An example of using **rgetdat** and **rputdat** for C can be located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\C\Napi\st** project.

An example of using **rgetdat\_xxxx** and **rputdat\_xxxx** for VB can be located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\Vb\Vbnetapi\st** project.

---

## Accessing a whole record

It is often the case that you will need to read an entire record. It is a good idea to write a function in order to do this so that your main program can deal with the record data in a structure rather than as single fields. The following example shows how to write such a function.

First you must define a structure that closely matches the user table. The record read function returns the record data in this format. It is called with: the name of the server, the record number, and the address of the structure to fill out. Note that the table number is not passed as it is implied by the function name.

Within the record read function, a static structure is initialized to match the layout of the user table record. This is the easiest way to set up the array of structures that need to be passed to the **rgetdat** call. It is also good practice to isolate this structure to this function by defining it as a static variable.

When the record read function is called, some minor parameter checking is performed then all the record numbers in the array are set to the required record. A call to **rgetdat** is made to read the individual fields. After this is checked, the individual field values are copied into the structure and passed back to the calling function.

A call to **rgetdat/rputdat** is not limited to retrieving only a handful of fields. For example a single call to **rgetdat** could retrieve up to 180 real8 fields. The actual limit to the number of fields can be determined by using:

```
(22 x number of fields) + sum all string lengths <4000
```

Therefore, one **rgetdat** call could retrieve multiple records which could improve efficiency and program execution time. The function above could be easily changed to do this.

### Sample record read function for C

```
/* C structure of user table 07 */
typedef struct tagUSTBL07
{
    int2        ipack;
    float       tmout;
    float       dout;
    float       bmax;
    float       cmin;
    float       bav;
    float       cav;
    float       idq;
} USTBL07;
```

---

```
/* retrieve a single record from user table 07 */
int rget_ustbl07(host,recno,rec)
char *host;      /* (in) host name of the system */
int recno;       /* (in) number of the record to retrieve */
USTBL07 *rec;    /* (out) record contents returned */
{
/* rgetdat structure of user table 07 */
static rgetdat_data ustbl07_def[]=
    {
/* type          file    rec  word  start len */
    {RGETDAT_TYPE_INT2,  UTBL07_F, 1,   1,   0,   0},
    {RGETDAT_TYPE_REAL4, UTBL07_F, 1,   2,   0,   0},
    {RGETDAT_TYPE_REAL4, UTBL07_F, 1,   4,   0,   0},
    {RGETDAT_TYPE_REAL4, UTBL07_F, 1,   6,   0,   0},
    {RGETDAT_TYPE_REAL4, UTBL07_F, 1,   8,   0,   0},
    {RGETDAT_TYPE_REAL4, UTBL07_F, 1,  10,   0,   0},
    {RGETDAT_TYPE_REAL4, UTBL07_F, 1,  12,   0,   0},
    {RGETDAT_TYPE_BITS,  UTBL07_F, 1,  14,   0,   8}
    };

#define USTBL07_FLDS sizeof(ustbl07_def)/sizeof(rgetdat_data)

int ierr;
int i;

/* validate the host name */
if (host==NULL)
    return 1;

/* validate the rec pointer */
if (rec==NULL)
    return 2;

/* set the record number to retrieve */
for (i=0; i<USTBL07_FLDS; i++)
    ustbl07_def[i].rec=recno;
```

---

---

```

    /* retrieve the actual record */
    if ((ierr=rgetdat(host,USTBL07_FLDS,ustbl07_def))!=0)
        return ierr;

    /* check the return status of each getdat call */
    for (i=0; i<USTBL07_FLDS; i++)
    {
        if (ustbl07_def[i].status!=0)
            return ustbl07_def[i].status;
    }

    /* copy the values retrieved into the structure */
    rec->ipack=ustbl07_def[0].value.int2;
    rec->tmout=ustbl07_def[1].value.real4;
    rec->dout=ustbl07_def[2].value.real4;
    rec->bmax=ustbl07_def[3].value.real4;
    rec->bav=ustbl07_def[4].value.real4;
    rec->cav=ustbl07_def[5].value.real4;
    rec->idq=ustbl07_def[6].value.bits;

    return 0;
}

```

---

### Sample record read function for VB

In this example user table, seven records contain eight floating point values.

```

'VB structure for user table 07
Private Type USTBL07
    ipack As Single
    tmout As Single
    dout As Single
    bmax As Single
    cmin As Single
    bav As Single
    cav As Single
    idq As Single
End Type

```

---

```
'file number for user table 07
Const UTBL07_F = 257

Private Function rget_ustbl07(ByVal host As String, ByVal recno
As Integer, rec As USTBL07) As Integer

    'declare data
    Dim ustbl07_def(7) As rgetdat_float_data_str

    'setup data
    For cnt = 0 To 7
        ustbl07_def(cnt).file = UTBL07_F
        ustbl07_def(cnt).rec = recno
        ustbl07_def(cnt).word = (cnt * 2) + 1
    Next

    'retrieve the actual record
    rget_ustbl07 = RGetDat_Float(host, 8, ustbl07_def)

    'check the return status
    If rget_ustbl07 <> 0 Then
        Exit Function
    End If

    'check the status on each value
    For cnt = 0 To 7
        If ustbl07_def(cnt).status <> 0 Then
            rget_ustbl07 = ustbl07_def(cnt).status
            Exit Function
        End If
    Next

    'copy the values retrieved into the structure
    rec.ipack = ustbl07_def(0).value
    rec.tout = ustbl07_def(1).value
```

---

```
rec.dout = ustbl07_def(2).value
rec.bmax = ustbl07_def(3).value
rec.cmin = ustbl07_def(4).value
rec.bav = ustbl07_def(5).value
rec.cav = ustbl07_def(6).value
rec.idq = ustbl07_def(7).value
```

End Function

This method could quite easily be applied for writing a whole record of a user table as well by using the **rputdat** function.

---

### Looking up error strings

All of the Network API for Windows functions return a non-zero value when they encounter a problem performing an operation. The value returned can be used to lookup an error string which describes the type of error that occurred. The function **hsc\_napierrstr** is used to lookup the error string from the error number.

---

#### Attention:

The hexadecimal return value '839A' (NADS\_PARTIAL\_FUNC\_FAIL) indicates that a partial function fail has occurred and is only a warning. This warning indicates that at least one request, and possibly all requests, made to a list-based function has failed. If this value is received, the **fstatus** Field of the data structure for each request should be checked for errors.

---

---

### Example

An example of using **hsc\_napierrstr** for C can be located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\C\Napitst** project.

You should use **hsc\_napierrstr2** for VB. **hsc\_napierrstr** is provided for backward compatibility only.

An example of using **hsc\_napierrstr2** for VB can be located in the **\<install folder>\Honeywell\Experion PKS\Client\Netapi\Samples\VB\Vbnetapitst** project.

---

There is a special condition for error numbers. If the lower four digits of the hexadecimal

error number is '8150', then the top four digits gives an Experion Process Controller error. In this case, `hsc_napierrstr` cannot be called to resolve the error number. Instead, you can look at the file `M4_err_def` in the include folder for the error string corresponding to the top four-digit Experion Process Controller error code.

---

### Example

Consider the return value: 0x01068150. The lower four digits (8150) indicates that this is an Experion Process Control Software error. The entry for 0106 in `M4_err_def` indicates that the error is due to a 'timeout waiting for response'.

---

### Functions for accessing parameter values by name

Access to parameter values by using point and parameter names is provided by the functions:

- `rhsc_param_value_bynames`
- `rhsc_param_value_put_bynames`
- `rhsc_param_hist_offset_bynames`
- `rhsc_param_hist_date_bynames`

By using these functions in your interface, you are able to make requests using point names and parameter names. The resolution of these names is handled by the server. Data is manipulated via a single Network API call.

Be aware, however, that these functions produce significantly lower system performance than manually storing the results of the `rhsc_point_numbers_2` and `rhsc_param_numbers_2` functions locally. This is for two reasons:

- First, the server needs to resolve point and parameter names to numbers every time the functions are called, rather than just once at the start of the application.
- Second, point and parameter names are generally significantly longer than point and parameter numbers so there is greater network traffic.

The function `rhsc_param_value_bynames` is equivalent to:

`rhsc_point_numbers_2 + rhsc_param_numbers_2 + rhsc_param_values_2`

The function `rhsc_param_value_put_bynames` is equivalent to:

`rhsc_point_numbers_2 + rhsc_param_numbers_2 + rhsc_param_value_puts_2`

The function `rhsc_param_hist_offset_bynames` is equivalent to:

`rhsc_point_numbers_2 + rhsc_param_numbers_2 + rhc_param_hist_offsets_2`

The function **rhsc\_param\_hist\_date\_bynames** is equivalent to:

**rhsc\_point\_numbers\_2 + rhsc\_param\_numbers\_2 + rhsc\_param\_hist\_dates\_2**

---

**Attention:**

Optimum performance can only be achieved by using the functions **rhsc\_point\_numbers\_2** and **rhsc\_param\_numbers\_2** to resolve point names and parameter names. The names are resolved just once, at the start of the program. The equivalent point numbers and parameters numbers are returned by these functions and should be stored locally so that all subsequent requests to parameter and history values use the locally stored numbers.

---

Although **rhsc\_param\_value\_put\_byname** is a list based function, there is no implication that it should be used as a sequential write function. If any individual put fails, the function will not prevent the remaining writes from occurring. The function will instead continue to write values to the remaining point parameters in the list.

Be careful when using **rhsc\_param\_value\_put\_bynames()** with more than one point/parameter pair. Each put causes a control to be executed on the server and each control takes a small amount of time. If more than one pair is put, the total time for each of these controls may exceed the default TCP/IP timeout. This will cause the Network API to report the error **RCV\_TIMEOUT**, even though all puts may have been successful. In addition, the Network API will be unavailable until the list of puts has been processed. This could cause subsequent calls to the network API to fail until the list is processed.

If maximum performance from the Network API is not a major consideration for your network application, then use the **rhsc\_param\_value\_bynames**, **rhsc\_param\_value\_put\_bynames**, **rhsc\_param\_hist\_offset\_bynames** and **rhsc\_param\_hist\_date\_bynames** functions.

## Using Microsoft Visual Studio or Visual Basic to develop Network API applications

If you are developing Network API applications in C/C++, you need Microsoft Visual Studio 2012.

If you are developing Network API applications in Visual Basic, you need Microsoft Visual Basic Version 6 SP5.

### Using the Visual Basic development environment

Visual Basic programs that need to call the Network API will need to add a reference to the Network API dll in the project reference.

To do this, select the **ProjectReferences**.

1. When the list of available references is displayed, click **Browse**.
2. Browse to the **Windows system32** directory, locate the **hscnetapi.dll** file, and click **Open**.
3. Click **OK** to save the information.

### Changing packing settings when compiling C++ applications

When using C++ in Visual Studio, certain settings that affect the interpretation of header files should *not* be changed from their defaults when compiling applications as it will cause the Experion header files to be interpreted incorrectly.

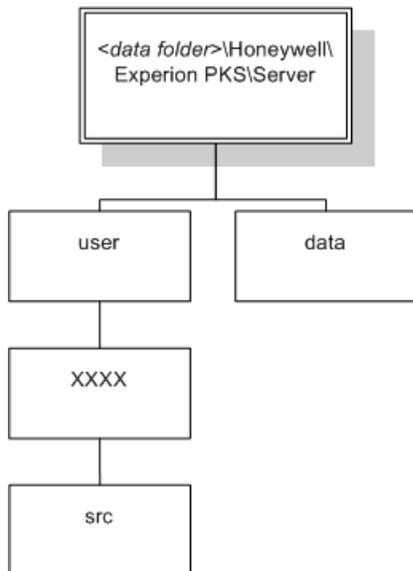
If you do need to change the packing setting, use **#pragma** lines instead to change the settings for your code but not for the Experion headers. For example, the following code is legitimate:

```
#include <Experion header>
#pragma pack(push, 2)
#include <Customer Code>
#pragma pop()
#include <More Experion headers>
```

### Folder structures for C/C++ applications

The folder structures shown below should be used for development of C/C++ Server API applications which will run on the server.

**<data folder>** server folder structure

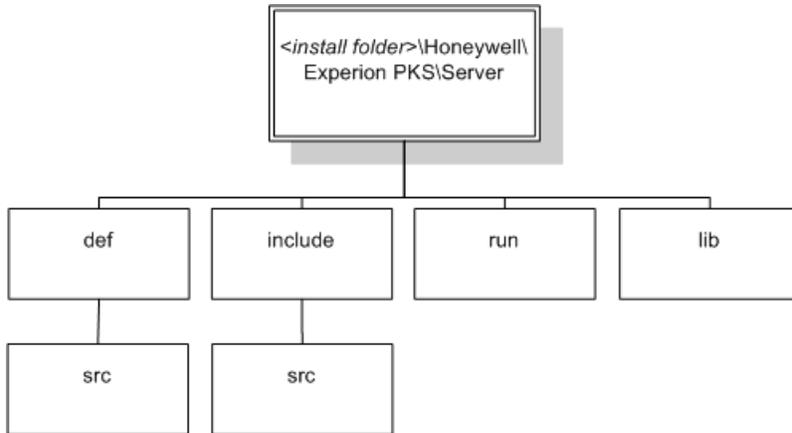


**<data folder>** is the location where Experion data is stored. For default installations, this is **C:\ProgramData**.

The **user/xxxx/src** folder contains all source code files and make files for a particular application. **xxxx** should be representative of the function of the application.

The **server/data** folder contains all server database files.

**<install folder>** server folder structure



**<install folder>** is the location where Experion is installed. For default installations this is **C:\Program Files (x86)**.

The **def** folder contains real time database definitions.

The **include** and **include/src** folders contain global server definitions, such as system constants or data arrays in the form of C/C++ include files.

The **run** folder contains all server programs (including applications). This folder is included in the path of any server user.

The **lib** folder contains libraries used for application development.

## Network API applications fail to run

If you have developed a NetAPI application and try to run it on a node that doesn't have Experion HS NetAPI installed, the application will fail to run. The **hscnetapi.dll** that is shipped with Experion HS NetAPI is required for all NetAPI applications to run on an Experion HS server.

To resolve this problem, copy the file **%windir%\System32\hscnetapi.dll** from a node that has the Experion HS NetAPI installed and copy it into the same directory on the computer that doesn't have Experion HS NetAPI installed and needs to run the application.

Network writes may fail if **Disable writes via Network API** has been selected. For more information, see “Security tab, server wide settings” in the *Server and Client Configuration Guide*. Note that during a clean installation of Experion, writes via the Network API will be disabled by default.

---

## Network API Function Reference

Network API is part of the Open Data Access (ODA) option. It allows you to create applications—in Visual C/C++ or Visual Basic—that exchange data with the Experion database. These applications can run on another computer or the Experion server.

Applications that use Network API can have:

- Read/write access to point parameter values
- Read access to history data
- Read/write access to server database files (user files)

The Network API provides the ability to remotely interrogate and change values in the server database through a set of library routines.

Functions that enable access to remote point history data and user tables are available as well as remote point control via a TCP/IP network.

There is one significant difference between Network API remote server functions and local Server functions. The Network API functions, where sensible, allow multiple invocations of the API function remotely using a single request through the use of list blocks passed as arguments. This enables network bandwidth and processing resources to be used more sparingly. In other respects, the functions closely follow the functionality of their standard Server API equivalents.

The following sections describe:

- *Functions* below
- *Backward-compatibility Functions* on page 341
- *Diagnostics for Network API functions* on page 375

### Functions

This section contains Network API functions.

### See also

*Backward-compatibility Functions* on page 341

*Diagnostics for Network API functions* on page 375

### **hsc\_bad\_value**

Checks whether the parameter value is bad.

### C/C++ Synopsis

```
int hsc_bad_value (float nValue)
```

### VB Synopsis

```
hsc_bad_value (ByVal nValue as Single) As Boolean
```

### Arguments

| Argument | Description              |
|----------|--------------------------|
| nValue   | (in) The parameter value |

### Description

This function is really only useful for the history functions, which do return bad values. Returns TRUE (-1) if the parameter value is BAD; otherwise FALSE (0).

### hsc\_napierrstr

Lookup an error string from an error number. This function is provided for backward compatibility.

### C/C++ Synopsis

```
void hsc_napierrstr(UINT err, LPSTR texterr);
```

### VB Synopsis

```
hsc_napierrstr(ByVal err As Integer) As String
```

### Arguments

| Argument | Description                     |
|----------|---------------------------------|
| err      | (in) The error number to lookup |
| texterr  | (out) The error string returned |

### Diagnostics

This function will always return a usable string value.

### rgetdat

Retrieve a list of fields from a user file.

### C/C++ Synopsis

```
int rgetdat(char *server,
            int num_points,
            rgetdat_data* getdat_data);
```

### VB Synopsis

```
rgetdat_int(ByVal server As String,
            ByVal num_points As Integer,
            getdat_int_data() As rgetdat_int_data_str) As Integer
rgetdat_bits(ByVal server As String,
            ByVal num_points As Integer,
            getdat_bits_data() As rgetdat_bits_data_str) As Integer
rgetdat_long(ByVal server As String,
            ByVal num_points As Integer,
            getdat_long_data() As rgetdat_long_data_str) As Integer
rgetdat_float(ByVal server As String,
            ByVal num_points As Integer,
            getdat_float_data() As rgetdat_float_data_str) As Integer
rgetdat_double(ByVal server As String,
            ByVal num_points As Integer,
            getdat_double_data()
            As rgetdat_double_data_str) As Integer
rgetdat_str(ByVal server As String,
            ByVal num_points As Integer,
            getdat_str_data()
            As rgetdat_str_data_str) As Integer
```

### Arguments

| Argument                | Description                                                                                      |
|-------------------------|--------------------------------------------------------------------------------------------------|
| <b>server</b>           | (in) Name of server that the database resides on.                                                |
| <b>num_points</b>       | (in) The number of points passed to <b>rgetdat_xxxx</b> in the <b>getdat_xxxx_data</b> argument. |
| <b>getdat_xxxx_data</b> | (in/out) Pointer to a series of <b>rgetdat_xxxx_data</b> structures (one for each point).        |

## Description

This function call enables fields from a user table to be accessed. The fields to be accessed are referenced by the members of the `rgetdat_data` structure (see below). The function accepts an array of **rgetdat\_data** structures thus providing the flexibility to obtain multiple fields with one call. Note that for the C interface only (not the VB interface), the fields can be of different types and from different user tables.

Note that a successful return status from the **rgetdat** call indicates that no network errors were encountered (that is, the request was received, processed and responded to). The status field in each call structure needs to be verified on return to determine the result of the individual remote calls.

The program using this function call must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network. To meet this requirement, adhere to the following guideline:

`(22 * number of fields) + sum of all string value lengths in bytes < 4000`

The structure of the **rgetdat\_data** structure is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                            |                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>int2 type</b>           | (in) Defines the type of data to be retrieved/stored, this will be one of the standard server data types. Namely using one of the following defines:<br>RGETDAT_TYPE_INT2, RGETDAT_TYPE_INT4, RGETDAT_TYPE_REAL4, RGETDAT_TYPE_REAL8, RGETDAT_TYPE_STR, RGETDAT_TYPE_BITS                                                                                                                   |
| <b>int2 file</b>           | (in) Absolute database file number to retrieve/store field.                                                                                                                                                                                                                                                                                                                                 |
| <b>int2 rec</b>            | (in) Record number in above file to retrieve/store field.                                                                                                                                                                                                                                                                                                                                   |
| <b>int2 word</b>           | (in) Word offset in above record to retrieve/store field.                                                                                                                                                                                                                                                                                                                                   |
| <b>int2 start_bit</b>      | (in) Start bit offset into the above word for the first bit of a bit sequence to be retrieved/stored. (That is, the bit sequence starts at: word + start_bit, where start_bit=0 is the first bit in the field.) Ignored if type is not a bit sequence.                                                                                                                                      |
| <b>int2 length</b>         | (in) Length of bit sequence or string to retrieve/store, in characters for a string, in bits for a bit sequence. Ignored if type is not a string or bit sequence.                                                                                                                                                                                                                           |
| <b>int2 flags</b>          | (in) Specifies the direction to read/write for circular files. (0: newest record, 1: oldest record)                                                                                                                                                                                                                                                                                         |
| <b>rgetdat_value value</b> | (in/out) Value of field retrieved or value to be stored. When storing strings they must be of the length given above. When strings are retrieved, they become NULL terminated, hence the length allocated to receive the string must be one more than the length specified above. Bit sequences will start at bit zero and be length bits long. See below a description of the union types. |

|                            |                                                                                                                                                                                                                                          |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>int2 status</b>         | (out) return value of actual remote getdat/putdat call.<br>The union structure of the value member used in the <b>rgetdat_data</b> structure is defined in <b>nif_types.h</b> . This structure, and its members, are defined as follows: |
| <b>short int2</b>          | Two byte signed integer.                                                                                                                                                                                                                 |
| <b>long int4</b>           | Four byte signed integer.                                                                                                                                                                                                                |
| <b>float real4</b>         | Four byte IEEE floating point number.                                                                                                                                                                                                    |
| <b>double real</b>         | Eight byte (double precision) IEEE floating point number.                                                                                                                                                                                |
| <b>char* str</b>           | Pointer to string. (Note allocation of space for retrieving a string is the responsibility of the program calling <b>rgetdat</b> , see <b>rgetdat_data</b> structure description above).                                                 |
| <b>unsigned short bits</b> | Two byte unsigned integer to be used for bit sequences (partial integer). Note the maximum length of a bit sequence is limited to 16.                                                                                                    |

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## rhsc\_notifications

Insert an alarm or event into the event log.

## C/C++ Synopsis

```
int rhsc_notifications(char *szHostname,
    int cjrnd,
    NOTIFICATION_DATA* notd);
```

## VB Synopsis

```
RHSC_notifications(ByVal hostname As String,
    ByVal num_requests As Long,
    notification_data_array()
    As notification_data) As Long
```

## Arguments

| Argument          | Description                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server that the data resides on (that is, server hostname)                           |
| <b>cprmbd</b>     | (in) Number of notifications requested                                                            |
| <b>notd</b>       | (in/out) Pointer to an array of NOTIFICATION_DATA structures (one array element for each request) |

## Description

The structure of the NOTIFICATION\_DATA structure is defined in netapi\_types.h. This structure and its members are defined as follows:

|                             |                                       |
|-----------------------------|---------------------------------------|
| <b>struct timeb timebuf</b> | (in) reserved for future use          |
| <b>n_long nPriority</b>     | (in) priority                         |
| <b>n_long nSubPriority</b>  | (in) sub priority                     |
| <b>n_char* szName</b>       | (in) name (usually pnt name)          |
| <b>n_char* szEvent</b>      | (in) event (eg. RCHANGE)              |
| <b>n_char* szAction</b>     | (in) action (eg. OK, ACK, CNF)        |
| <b>n_char* szLevel</b>      | (in) level (eg. CB, MsEDE, NAPI)      |
| <b>n_char* szDesc</b>       | (in) description (usually param name) |
| <b>n_char* szValue</b>      | (in) alarm value                      |
| <b>n_char* szUnits</b>      | (in) alarm units                      |
| <b>n_long fStatus</b>       | (out) unused at the moment            |

RHSC\_NOTIFICATIONS can be used to remotely generate alarms and events. The various text fields are the raw data that can be specified. Not all the fields are applicable to every type of notification. The data in these fields are formatted for you into a standard event log line on the server. The **nPriority** field is used to define the behavior on the server. The following constants are defined in **nads\_def.h**:

```

NTFN_ALARM_URGENT    generates an urgent
level alarm
NTFN_ALARM_HIGH      generates a high level alarm
NTFN_ALARM_LOW       generates a low level alarm
NTFN_ALARM_JNL       generates a journal level
alarm
    
```

```

NTFN_EVENT          only generates an event
                    (nothing
will be logged to the
                    alarm
list)

```

A number of predefined strings have been provided for use in the **szEvent**, **szAction** and **szLevel** fields. Although there is no requirement to use these strings, their use will promote consistency. They can be found in **nads\_def.h**.

```

static char* EventStrings[] =
{
    // should be an alarm type, an event type from
    // one of the following strings, blank, or user defined
    'CHANGE',      // local operator change
    'ACHANGE',     // application (non-Station) change
    'LOGIN',       // operator login
    'ALOGIN',      // application (non-Station) login
    'WDT',         // watch dog timer event
    'FAILED'       // operation failed
};

static char* ActionStrings[] =
{
    // should be blank (new alarm), an event type from
    // one of the following strings or user defined
    'OK',          // alarm returned to normal
    'ACK',         // alarm acknowledged
    'CNF'          // message confirmed
};

static char* LevelStrings[] =
{
    // should be an alarm level, one of the
    // following strings indicating where the event
    // was generated from, blank, or user defined
    'CB',          // Control Builder
    'MSEDE',       // Microsoft Excel Data Exchange
    'NAPI',        // Network API application
    'API'          // API application
};

```

A successful return status from the **rhsc\_notifications** call indicates that no network errors were encountered (that is, the request was received, processed, and responded to). If the returned value is `NADS_PARTIAL_FUNC_FAIL`, then at least one request (and possibly all requests) failed. The status field of each array element should be checked to find which request failed.

It is the responsibility of the program using this function call to ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network. To meet this requirement, adhere to the following guideline:

- For ALL request packets:

```
(15 * number of notifications) + sum of all string lengths < 4000
```

Note that the sum of the string lengths does not include nulls, which is the convention for C/C++.

---

### Example

Create an event log entry indicating that a remote control has just occurred.

```
#include <sys/timeb.h>
int status;
int i;
/* Set the point names, parameter names and parameter offsets to appropriate values */
PARAM_DATA rgprmbd[] = {{'pntanal','DESC', 1}};
#define cprmbd (sizeof(rgprmbd)/sizeof(PARAM_DATA))
/* Setup parameters for the call to rhsc_notifications */
NOTIFICATION_DATA rgnotd[cprmbd];

/* Allocate space and set the value and type to control pntanal.SP to 42.0 */
rgprmbd[0].pupvValue = (PARvalue *)malloc(sizeof (PARvalue));
strcpy(rgprmbd[0].pupvValue->text, 'FunkyDescription');
rgprmbd[0].nType = DT_CHAR;

/* Set up the rest of the parameters for the notification */
ftime(&rgnotd[0].timebuf); /* Set the time */
rgnotd[0].nPriority = NTFN_EVENT; /* Event only */
rgnotd[0].nSubPriority = 0; /* Subpriority */
rgnotd[0].szName = rgprmbd[0].szPntName; /* Set the point name */
*/
```

---

---

```

rgnotd[0].szEvent = EventStrings[0];           /* Assign event to be
'RCHANGE' */
rgnotd[0].szAction = ActionStrings[0];        /* Action is 'OK' */
rgnotd[0].szLevel = LevelStrings[2];         /* Notification from
'NAPI' */
rgnotd[0].szDesc = rgprmbd[0].szPrmName;     /* Parameter name */
rgnotd[0].szValue = rgprmbd[0].pupvValue->text; /* Value to control
'SP' to */
rgnotd[0].szUnits = '';                      /* No units */
status = rhsc_param_value_put_bynames('server1', cprmbd, rgprmbd);

/* The notification is created here! */
status = rhsc_notifications('server1', cprmbd, rgnotd);

```

---

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## See also

*hsc\_notif\_send()* on page 146

### rhsc\_param\_hist\_date\_bynames

Retrieve history values for Parameters referenced by name from a start date.

This function's synopsis and description is identical to that of 'rhsc\_param\_hist\_offset\_bynames.'

### rhsc\_param\_hist\_offset\_bynames

Retrieve history values for parameters referenced by name from an offset.

## C Synopsis

```

int rhsc_param_hist_date_bynames(char
*szHostName,
    int                cHstRequests,
    HIST_BYNAME_DATA* rghstbd);

int rhsc_param_hist_offset_bynames(char *szHostName,

```

```
int cHstRequests,
HIST_BYNAME_DATA* rghstbd);
```

### VB Synopsis

```
RHSC_param_hst_date_bynames (ByVal
Server As String,
    ByVal num_requests As Long,
    hist_byname_data_array()
    As hist_byname_data) As Long
```

```
RHSC_param_hst_offset_bynames (ByVal
Server As String,
    ByVal num_requests As Long,
    hist_byname_data_array()
    As hist_byname_data) As Long
```

### Arguments

| Argument            | Description                                                                                        |
|---------------------|----------------------------------------------------------------------------------------------------|
| <b>szHostName</b>   | (in) Name of server on which the data resides                                                      |
| <b>cHstRequests</b> | (in) Number of rghstbd elements                                                                    |
| <b>rghstbd</b>      | (in / out) Pointer to an array of HIST_BYNAME_DATA structures. One array element for each request. |

### Description

The structure of the HIST\_BYNAMES\_DATA structure is defined in **netapi\_types.h**. This structure and its members are defined as follows:

|                                 |                                                                                                                                                                                                  |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>n_long dtStartDate</b>       | (in) The start date of history to retrieve in Julian days (number of days since 1 Jan 1981). If the function called is rhsc_param_hist_offset_bynames then this value is ignored.                |
| <b>n_float tmStartTime,</b>     | (in) The start time of history to retrieve in seconds since midnight. If the function called is rhsc_param_hist_offset_bynames then this value is ignored.                                       |
| <b>n_long nHstOffset,</b>       | (in) Offset from latest history value in history intervals (where offset=1 is the most recent history value). If the function called is rhsc_param_hist_date_bynames then this value is ignored. |
| <b>n_long fGetHstParStatus,</b> | (out) The status returned by the gethstpar function.                                                                                                                                             |

|                                     |                                                                                                                                                                                                                                                                  |
|-------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>n_short<br/>nHstType,</b>        | (in) The type of history to retrieve (See Description).                                                                                                                                                                                                          |
| <b>n_ushort<br/>cPntPrmNames,</b>   | (in) The number of point / parameter pairs requested.                                                                                                                                                                                                            |
| <b>n_ushort<br/>cHstValues</b>      | (in) The number of history values to be returned per point / parameter pair. This value must not be negative: the error message 'Message being built too large' is returned if it is.                                                                            |
| <b>n_char*<br/>szArchivePath,</b>   | This member is no longer in use and is only retained for backwards compatibility. Instead, pass a zero length string.                                                                                                                                            |
| <b>n_char**<br/>rgszPointNames,</b> | (in) An array of point names to process.                                                                                                                                                                                                                         |
| <b>n_char**<br/>rgszParamNames</b>  | (in) An array of parameter names to process. Each parameter is associated with the corresponding entry in the rgszPointNames array.                                                                                                                              |
| <b>n_long*<br/>rgfPntPrmStatus</b>  | (out) The status returned by the Server when calling <b>hsc_point_number</b> and <b>hsc_param_number</b> for each point parameter pair.                                                                                                                          |
| <b>n_float*<br/>rgnHstValues</b>    | (out) A pointer to the list of returned history values. The history values are stored in rHstValuessized blocks for each point parameter pair. If there is no history for the requested time or if the data was bad, then the value -0.0 is stored in the array. |

These functions request a number of blocks of history data from a remote server. For each block, a history type is specified using one of the following values:

| Value       | Description                       |
|-------------|-----------------------------------|
| HST_1MIN    | One minute Standard history       |
| HST_6MIN    | Six minute Standard history       |
| HST_1HOUR   | One hour Standard history         |
| HST_8HOUR   | Eight hour Standard history       |
| HST_24HOUR  | Twenty four hour Standard history |
| HST_5SECF   | Fast history                      |
| HST_1HOURE  | One hour Extended history         |
| HST_8HOURE  | Eight hour Extended history       |
| HST_24HOURE | Twenty four hour Extended history |

Depending upon which function is called, history will be retrieved from a specified date and time or offset going backwards in time. The number of history values to be retrieved per point is specified by **cHstValues**. **cHstValues** must not be negative. Point parameters are specified by name only and all name to number resolutions are performed by the server.

Before making a request you must allocate sufficient memory for each list pointed to by **rgnHstValues**. You must also free this memory before exiting your network application. The number of bytes required for each request is  $4 * \text{cHstValues} * \text{cPntPrmNames}$ .

A successful return status from the **rhsc\_param\_hist\_xxxx\_bynames** call indicates that no network errors were encountered (that is, the request was received, processed, and responded to). If the returned value is **NADS\_PARTIAL\_FUNC\_FAIL**, then at least one request (and possibly all requests) failed. The status fields of each array element should be checked to find which request failed.

The program using this function call must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network.

To meet the program requirement not to exceed the maximum packet size permitted, adhere to the following guidelines:

- For request packets for **rhsc\_param\_hist\_date\_bynames**:

$(15 * \text{number of history requests}) + (2 * \text{number of point parameter pairs})$   
 + sum of string lengths of point names, parameter names and archive paths  
 in bytes < 4000

- For request packets for **rhsc\_param\_hist\_offset\_bynames**:

$(11 * \text{number of history requests}) + (2 * \text{number of point parameter pairs})$   
 + sum of string lengths of point names, parameter names and archive paths  
 in bytes < 4000

- For response packets:

$(4 * \text{number of history requests}) + (4 * (\text{For each history request the sum of } (\text{cPntPrmNames} + \text{cPntPrmNames} * \text{cHstValues}))) < 4000$

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

---

### Example

For **rhsc\_param\_hist\_date\_bynames**

---

---

```
int status;
int i;
n_long ConvertToDays(n_long year, n_long month, n_long day)
{
    n_long nConvertedDays = 0;
    n_long leap = 0;
    nConvertedDays = (year - 1981) * 365 + (year - 1981) / 4 + day - 1;
    leap = ((year % 400) == 0) || (((year % 100) != 0) && ((year % 4) ==
0));
    switch(month)
    {
    case 12:
        nConvertedDays += 30;
    case 11:
        nConvertedDays += 31;
    case 10:
        nConvertedDays += 30;
    case 9:
        nConvertedDays += 31;
    case 8:
        nConvertedDays += 31;
    case 7:
        nConvertedDays += 30;
    case 6:
        nConvertedDays += 31;
    case 5:
        nConvertedDays += 30;
    case 4:
        nConvertedDays += 31;
    case 3:
        nConvertedDays += 28 + leap;
    case 2:
        nConvertedDays += 31;
    case 1:
        break;
    default:
        printf("Invalid month\n");
    }
}
```

---

---

```
        return 0;
    }
    return nConvertedDays;
}

n_float ConvertToSeconds(int hour, int minute, int second)
{
    return (n_float)(second + (minute * 60) + (hour * 3600));
}

int main()
{
    HIST_BYNAME_DATA rghstbd[2];
    int chstbd = 2;
    /* Set up date and time for 7 November 2001 at 1:00 pm */
    n_long year = 2001;
    n_long month = 11;
    n_long day = 7;
    int hour = 13;
    int minute = 0;
    int second = 0;

    /* Allocate memory and set up rghstbd */
    for (i=0; i<chstbd; i++)
    {
        rghstbd[i].dtStartDate = ConvertToDays(year,month,day);
        rghstbd[i].tmStartTime = ConvertToSeconds(hour,minute,second);
        rghstbd[i].nHstType = HST_5SECF;
        rghstbd[i].cPntPrmNames = 3;
        /* Two point parameter pairs */
        rghstbd[i].cHstValues = 10;
        /* Ten history values each */
        rghstbd[i].szArchivePath = "ay2001m11d01h13r001";
        rghstbd[i].rgszPointNames = (char **)malloc(sizeof(char *) * 3);
        rghstbd[i].rgszPointNames[0]="AnalogPoint";
        rghstbd[i].rgszPointNames[1]="AnalogPoint";
        rghstbd[i].rgszPointNames[2]="AnalogPoint";
    }
}
```

---

---

```

        rghstbd[i].rgszParamNames = (char **)malloc(sizeof(char *) * 3);
        rghstbd[i].rgszParamNames[0]="pv";
        rghstbd[i].rgszParamNames[1]="sp";
        rghstbd[i].rgszParamNames[2]="op";
        rghstbd[i].rgfPntPrmStatus = (n_long *)malloc(sizeof(n_long) * 3);
        rghstbd[i].rgnHstValues = (n_float *)malloc(sizeof(n_float) * 30);
    }

    status = rhsc_param_hist_date_bynames("Server1", chstbd, rghstbd);

    switch (status)
    {
    case 0:
        printf("rhsc_param_hist_date_bynames successful\n");
        /* Now print the 4th history value returned for AnalogPoint's op */
        printf("Value = %f\n",
            rghstbd[0].rgnHstValues[3 + rghstbd[0].chHstValues * 2]);
        break;
    case NADS_PARTIAL_FUNC_FAIL:
        printf("rhsc_param_hist_date_bynames partially failed");
        /* Check fStatus flags to find out which one(s) failed. */
        break;
    default:
        printf("rhsc_param_hist_date_bynames failed (c_geterrno() = 0x%x)",
            status);
        break;
    }

    for (i=0; i<chstbd; i++)
    {
        free(rghstbd[i].rgszPointNames);
        free(rghstbd[i].rgszParamNames);
        free(rghstbd[i].rgfPntPrmStatus);
        free(rghstbd[i].rgnHstValues);
    }
    return 0;
}

```

**For rhsc\_param\_hist\_offset\_bynames**

---

---

```
int status;
int i;
int main()
{
    HIST_BYNAME_DATA rghstbd[2];
    int chstbd = 2;
    n_long nOffset = 1;      /* Most recent history value */
    /* Allocate memory and set up rghstbd */
    for (i=0; i<chstbd; i++)
    {
        rghstbd[i].nHstOffset = nOffset;
        rghstbd[i].nHstType = HST_5SECF;
        rghstbd[i].cPntPrmNames = 3;
    /* Two point parameter pairs */
        rghstbd[i].cHstValues = 10;
    /* Ten history values each */
        rghstbd[i].szArchivePath = "ay2001m11d01h13r001";
        rghstbd[i].rgszPointNames = (char **)malloc(sizeof(char *) * 3);
        rghstbd[i].rgszPointNames[0]="AnalogPoint";
        rghstbd[i].rgszPointNames[1]="AnalogPoint";
        rghstbd[i].rgszPointNames[2]="AnalogPoint";
        rghstbd[i].rgszParamNames = (char **)malloc(sizeof(char *) * 3);
        rghstbd[i].rgszParamNames[0]="pv";
        rghstbd[i].rgszParamNames[1]="sp";
        rghstbd[i].rgszParamNames[2]="op";
        rghstbd[i].rgfPntPrmStatus = (n_long *)malloc(sizeof(n_long) * 3);
        rghstbd[i].rgnHstValues = (n_float *)malloc(sizeof(n_float) * 30);
    }

    status = rhsc_param_hist_offset_bynames("Server1", chstbd, rghstbd);

    switch (status)
    {
    case 0:
        printf("rhsc_param_hist_offset_bynames successful\n");
        /* Now print the 4th history value returned for AnalogPoint's op */
        printf("Value = %f\n",
```

---

---

```

        rghstbd[0].rgnHstValues[3 + rghstbd[0].cHstValues * 2]);
        break;
    case NADS_PARTIAL_FUNC_FAIL:
        printf("rhsc_param_hist_offset_bynames partially failed");
        /* Check fStatus flags to find out which one(s) failed. */
        break;
    default:
        printf("rhsc_param_hist_offset_bynames failed (c_geterrno() =
0x%x)", status);
        break;
    }

    for (i=0; i<chstbd; i++)
    {
        free(rghstbd[i].rgszPointNames);
        free(rghstbd[i].rgszParamNames);
        free(rghstbd[i].rgfPntPrmStatus);
        free(rghstbd[i].rgnHstValues);
    }
    return 0;
}

```

---

### **rhsc\_param\_hist\_dates\_2**

Retrieve history values for a point based on date.

This function's synopsis and description are identical to that of 'rhsc\_param\_hist\_offsets\_2.'

### **rhsc\_param\_hist\_offsets\_2**

Retrieve history values for a point based on offset.

### **C/C++ Synopsis**

```

    int rhsc_param_hist_dates_2
(
    char*    server,
    int     num_gethsts,
    rgethstpar_date_data_2* gethstpar_date_data2
);

```

```
int rhsc_param_hist_offsets_2
(
    char*    server,
    int      num_gethsts,
    rgethstpar_ofst_data_2* gethstpar_ofst_data2
);
```

### VB Synopsis

```
rHsc_Param_Hist_Dates_2(ByVal Server As String,
    num_requests As Long,
    gethstpar_date_data_array()
    As Hist_Value_Data_2) As Long
rHsc_Param_Hist_Offsets_2(ByVal Server As String,
    num_requests As Long,
    gethstpar_ofst_data_array()
    As Hist_Value_Data_2) As Long
```

### Arguments

| Argument                | Description                                                                                         |
|-------------------------|-----------------------------------------------------------------------------------------------------|
| <b>server</b>           | (in) Name of server that the database resides on                                                    |
| <b>num_requests</b>     | (in) The number of history requests                                                                 |
| <b>gethstpar_x_data</b> | (in/out) Pointer to an array of rgethstpar_x_data_2 structures (one array element for each request) |

### Description

rhsc\_param\_hist\_x\_2 functions are the replacements for the now deprecated rhsc\_param\_hist\_x functions.

---

**Attention:**

rhsc\_param\_hist\_x functions can only access points in the range: 1 <= point number <= 65,000.

---

Use this function to retrieve history values for points. The two types of history (based on time or offset) are retrieved using the corresponding function variation. History will be retrieved from a specified time or offset going backwards in time. The history values to be accessed are

referenced by the `rgethst_date_data_2` and `rgethst_ofst_data_2` structures (see below). The functions accept an array of these structures, thus providing access to multiple point history values with one function call.

Note that a successful return status from the `rhsc_param_hist_dates_2` or `rhsc_param_hist_offsets_2` calls indicates that no network errors were encountered (that is, the request was received, processed and responded to). The status field in each call structure needs to be verified on return to determine the result of the individual remote calls.

The structure of the `rgethstpar_date_data_2` struct is defined in **netapi\_types.h**. This structure and its members are defined as follows:

|                                               |                                                                                                                                                                                                                                                     |
|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>n_ushort<br/>hist_<br/>type</b>            | (in) Defines the type of history to retrieve, this will be one of the standard server history types. Namely using one of the following:<br><br>HST_1MIN, HST_6MIN, HST_1HOUR, HST_8HOUR, HST_24HOUR, HST_5SECF, HST_1HOURE, HST_8HOURE, HST_24HOURE |
| <b>n_ulong<br/>hist_<br/>start_<br/>date</b>  | (in) Start date of history to receive in Julian days (number of days since 1st January 1981).                                                                                                                                                       |
| <b>n_ufloat<br/>hist_<br/>start_<br/>time</b> | (in) Start time of history to retrieve in seconds since midnight.                                                                                                                                                                                   |
| <b>n_ushort<br/>num_<br/>hist</b>             | (in) Number of history values per point to be retrieved.                                                                                                                                                                                            |
| <b>n_ushort<br/>num_<br/>points</b>           | (in) Number of points to be processed. MAXIMUM value allowed is 20.                                                                                                                                                                                 |
| <b>n_ulong*<br/>point_<br/>type_<br/>nums</b> | (in) Array (of size <b>num_points</b> ) containing the point type/numbers of the point history values to retrieve.                                                                                                                                  |
| <b>n_ushort*<br/>point_<br/>params</b>        | (in) Array of (of size <b>num_points</b> ) containing the parameter numbers of the history values to retrieve.                                                                                                                                      |
| <b>n_char*<br/>archive</b>                    | (in) Pointer to the NULL terminated string containing the archive path name of the archive file. A NULL pointer implies that the system will use current history and any                                                                            |

|                               |                                                                                                         |
|-------------------------------|---------------------------------------------------------------------------------------------------------|
| <b>path</b>                   | archive files which correspond to the value of the date and time parameters.                            |
| <b>n_ufloat* hist_values</b>  | (out) Array (of size <b>num_points * num_hist</b> ) to provide storage for the returned history values. |
| <b>n_ushort gethst_status</b> | (out) Return value of the actual remote hsc_history call.                                               |

The structure of the `rgethstpar_ofst_data_2` struct is defined in `netapi_types.h`. This structure and its members are defined as follows:

|                                 |                                                                                                                                                                                                                                                         |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>n_ushort hist_type</b>       | (in) Defines the type of history to retrieve, this will be one of the standard server history types. Namely using one of the following defines:<br>HST_1MIN, HST_6MIN, HST_1HOUR, HST_8HOUR, HST_24HOUR, HST_5SECF, HST_1HOURE, HST_8HOURE, HST_24HOURE |
| <b>n_ushort hist_offset</b>     | (in) Offset from latest history value in history intervals where offset=1 is the most recent history value).                                                                                                                                            |
| <b>n_ushort num_hist</b>        | (in) Number of history values per point to be retrieved.                                                                                                                                                                                                |
| <b>n_ushort num_points</b>      | (in) Number of points to be processed. MAXIMUM value allowed is 20.                                                                                                                                                                                     |
| <b>n_ulong* point_type_nums</b> | (in) Array (of size <b>num_points</b> ) containing the point type/numbers of the point history values to retrieve.                                                                                                                                      |
| <b>n_ushort* point_params</b>   | (in) Array (of size <b>num_points</b> ) containing the parameter numbers of the history values to retrieve.                                                                                                                                             |
| <b>n_char* archive_path</b>     | (in) Pointer to the NULL terminated string containing the archive path name of the archive file. A NULL pointer implies that the system will use current history and any archive files which correspond to the value of the date and time parameters.   |

|                                        |                                                                                                         |
|----------------------------------------|---------------------------------------------------------------------------------------------------------|
| <b>n_float*<br/>hist_<br/>values</b>   | (out) Array (of size <b>num_points * num_hist</b> ) to provide storage for the returned history values. |
| <b>n_ushort<br/>gethst_<br/>status</b> | (out) Return value of the actual remote hsc_history call.                                               |

The program using this function call must ensure that the size of the network packets generated does not exceed the maximum packet size permitted on the network. To meet this requirement, adhere to the following guideline:

- For request packets for rhsc\_param\_hist\_dates\_2:

$$(16 * \text{number of rgethstpar\_date\_data\_2 structs}) + (6 * \text{combined point / param pair count across all rgethstpar\_date\_data\_2 structs}) + \text{combined string lengths of archive paths} < 4000.$$


---

**Attention:**

Combined point / parameter pair count across all rgethstpar\_x\_data\_2 structs may vary. For example, an rhsc\_param\_hist\_x\_2 call is made with 3 rgethstpar\_x\_data\_2 data structures, each referencing 2, 5 and 4 point / parameter pairs. This results in  $6*2 + 6*5 + 6*4 = 66$ , so:  $(16 * 3) + (6 * (2 + 5 + 4)) + (\text{strings ?}) < 4000$

---

- For request packets for rhsc\_param\_hist\_offsets\_2:

$$(12 * \text{number of rgethstpar\_ofst\_data\_2 structs}) + (6 * \text{combined point / param pair count across all rgethstpar\_ofst\_data\_2 structs}) + \text{combined string lengths of archive paths} < 4000.$$


---

**Attention:**

Combined point / parameter pair count across all rgethstpar\_x\_data\_2 structs may vary. For example, an rhsc\_param\_hist\_x\_2 call is made with 3 rgethstpar\_x\_data\_2 data structures, each referencing 2, 5 and 4 point / parameter pairs. This results in  $6*2 + 6*5 + 6*4 = 66$ , so:  $(16 * 3) + (6 * (2 + 5 + 4)) + (\text{strings ?}) < 4000$

---

- For response packets:

(4 \* number of history requests) + (4 \* (For each history request (num\_points \* num\_hist) ) < 4000

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

### rhsc\_param\_numbers\_2

Resolve a list of parameter names to numbers.

## C Synopsis

```
int rhsc_param_numbers_2
(
    char*    szHostname,
    int      cprmnd,
    PARAM_NUMBER_DATA_2* rgprmnd2
);
```

## VB Synopsis

```
rhsc_param_numbers_2 (ByVal
hostname As String,
    ByVal num_requests As Long,
    param_number_data_array()
As
    param_number_data_2)
As Long
```

## Arguments

| Argument          | Description                                                                                   |
|-------------------|-----------------------------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server on which the database resides                                             |
| <b>cprmnd</b>     | (in) The number of parameter name resolutions requested                                       |
| <b>rgprmnd2</b>   | (in/out) Pointer to an array of PARAM_NUMBER_DATA_2 structures (one for each point parameter) |

## Description

rhsc\_param\_numbers\_2 is the replacement for the now deprecated rhsc\_param\_numbers.

**Attention:**

rhsc\_param\_numbers can only access points in the range: 1 <= point number <= 65,000.

The structure of the PARAM\_NUMBER\_DATA\_2 structure is defined in netapi\_types.h. This structure and its members are defined as follows:

|                          |                                 |
|--------------------------|---------------------------------|
| <b>n_ulong nPnt</b>      | (in) point number               |
| <b>n_char* szPrmName</b> | (in) parameter name to resolve  |
| <b>n_ushort nPrm</b>     | (out) parameter number returned |
| <b>n_long fStatus</b>    | (out) status of each request    |

RHSC\_PARAM\_NUMBERS\_2 converts a list of point parameter names to their equivalent parameter numbers for a specified remote server.

A successful return status from the rhsc\_param\_numbers\_2 call indicates that no network errors were encountered (that is, the request was received, processed, and responded to). If the returned value is NADS\_PARTIAL\_FUNC\_FAIL, then at least one request (and possibly all requests) failed. The status field of each array element should be checked to find which request failed.

The program using this function call must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network. To meet this requirement, adhere to the following guidelines:

- For request packets:

```
(6 * number of points) + sum of string lengths of point names in bytes < 4000
```

- For response packets:

```
(8 * number of points) < 4000
```

**Example**

Resolve the parameter names 'pntana1.SP' and 'pntana1.DESC'.

```
int    status;
int    i;
POINT_NUMBER_DATA_2 rgpntnd2[] = {"pntana1"};
```

---

```
PARAM_NUMBER_DATA_2 rgprmnd2[] = {{0,
"SP"},{0, "DESC"}};

#define cpntnd sizeof(rgpntnd2)/sizeof(POINT_NUMBER_DATA_2)
#define cprmnd sizeof(rgprmnd2)/sizeof(PARAM_NUMBER_DATA_2)

status = rhsc_point_numbers_2("server1",
cpntnd, rgpntnd2);
/* Check for error status. */

/* Grab the point numbers from the rgpntnd2 array. */
Rgprmnd2[0].nPnt = rgpntnd2[0].nPnt;
Rgprmnd2[1].nPnt = rgpntnd2[0].nPnt;

status = rhsc_param_numbers_2("server1",
cprmnd, rgprmnd2);
switch (status)
{
    case 0:
        printf("rhsc_param_numbers_2
successful\n");
        for (i=0; i<cprmnd; i++)
        {
            printf("%s.%s has the parameter number %d\n",
                rgpntnd2[0].szPntName,
                rgprmnd2[i].szPrmName,
                rgprmnd2[i].nPrm);
        }
    case NADS_PARTIAL_FUNC_FAIL:
        printf("rhsc_param_numbers_2 partially failed\n");
        /* Check fStatus flags to find out which ones failed. */
        break;
    default:
        printf("rhsc_param_numbers_2 failed (c_geterrno() = 0x%x)\n",
            status);
        break;
}
```

---

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## See also

*rhsc\_point\_numbers\_2* on page 336

### rhsc\_param\_value\_bynames

Reads a list of point parameter values referenced by name.

## C/C++ Synopsis

```
int rhsc_param_value_bynames
(
    char*    szHostname,
    int      nPeriod,
    int      cprmbd,
    PARAM_BYNAME_DATA*    rgprmbd
);
```

## VB Synopsis

```
RHSC_param_value_bynames(ByVal hostname As String,
    ByVal period As Long,
    ByVal num_requests As Long,
    param_byname_data_array() As
    param_byname_data) As Long
```

## Arguments

| Argument          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server that the data resides on.                                                                                                                                                                                                                                                                                                                                                                                               |
| <b>nPeriod</b>    | (in) subscription period in milliseconds for the point parameters. Use the constant NADS_READ_CACHE if subscription is not required. If the value is in the Experion cache, then that value will be returned. Otherwise the controller will be polled for the latest value. Use the constant NADS_READ_DEVICE if you want to force Experion to re-poll the controller. The subscription period will not be applied to standard point types. |
| <b>cprmbd</b>     | (in) Number of parameter values requested.                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>rgprmbd</b>    | (in/out) Pointer to an array of PARAM_BYNAME_DATA structures (one array element for each request).                                                                                                                                                                                                                                                                                                                                          |

## Description

The structure of the `PARAM_BYNAME_DATA` structure is defined in `nif_types.h`. This structure and its members are defined as follows:

|                            |                                   |
|----------------------------|-----------------------------------|
| <b>n_char* szPntName</b>   | (in) point name                   |
| <b>n_char* szPrmName</b>   | (in) parameter name               |
| <b>n_long nPrmOffset</b>   | (in) parameter offset             |
| <b>PARvalue* pupvValue</b> | (out) parameter value union       |
| <b>n_ushort nType</b>      | (out) value type                  |
| <b>n_long fStatus</b>      | (out) status of each value access |

If your system uses *dynamic scanning*, `rhsc_param_value_bynames` calls from the Network API do not trigger dynamic scanning.

`RHSC_PARAM_VALUE_BYNAMES` requests a list of point parameter values from the specified remote server. Point parameters are requested by name only, and all name to number resolutions are performed by the server.

You can read a list of parameter values with different types using a single request. Each point parameter value is placed into a union (of type `PARvalue`). Before making the request, you must allocate sufficient memory for each value union. You must free this memory before exiting your network application.

A successful return status from the `rhsc_param_value_bynames` call indicates that no network errors were encountered (that is, the request was received, processed, and responded to). If the returned value is `NADS_PARTIAL_FUNC_FAIL`, then at least one request (and possibly all requests) failed. The status field of each array element should be checked to find which request failed.

The program using this function call must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network.

Due to the ability to acquire a list of parameters of mixed data type, it is difficult to give generic limits. To meet the program requirement not to exceed the maximum packet size permitted, adhere to the following guidelines given for a number of specific cases:

- For all request packets:

$(6 * \text{number of point parameters}) + \text{sum of string lengths of point names} + \text{sum of string lengths of parameter names} < 4000$

- For response packets when reading `DT_INT2` data only:

$(8 * \text{number of points parameters}) < 4000$

- For response packets when reading DT\_INT4 data only:  
(10 \* number of points parameters) < 4000
- For response packets when reading DT\_REAL data only:  
(14 \* number of points parameters) < 4000
- For response packets when reading DT\_DBLE data only:  
(14 \* number of points parameters) < 4000
- For response packets when reading DT\_CHAR data only:  
(7 \* number of points parameters) + sum of string lengths of value character strings in bytes < 4000
- For response packets when reading DT\_ENUM data only:  
(11 \* number of points parameters) + sum of string lengths of value enumeration strings in bytes < 4000

### Example

Read the value of pntana1.SP and pntana1.DESC.

```
int status;
int i;
PARAM_BYNAME_DATA rgprmbd[] = {{'pntana1','SP', 1},{'pntana1', 'DESC',
1}};
#define cprmbd (sizeof(rgprmbd)/sizeof(PARAM_BYNAME_DATA))

/* Allocate sufficient memory for each value union. See sample code
for rhsc_param_values for more details */
for (i=0; i<cprmbd; i++)
{
    rgprmbd[i].pupvValue = (PARvalue *)malloc(sizeof(PARvalue));
}

status = rhsc_param_value_bynames('server1', NADS_READ_CACHE, cprmbd,
rgprmbd);

switch (status)
{
    case 0:
```

---

```
printf('rhsc_param_value_bynames successful\n');
for (i=0; i<cprmbd; i++)
{
    switch (rgprmbd[i].nType)
    {
        case DT_CHAR:
            printf('%s.%s has the value %s\n',
                rgprmbd[i].szPntName,
                rgprmbd[i].szPrmName,
                rgprmbd[i].pupvValue->text);
            break;
        case DT_INT2:
            printf('%s.%s has the value %d\n',
                rgprmbd[i].szPntName,
                rgprmbd[i].szPrmName,
                rgprmbd[i].pupvValue->int2);
            break;
        case DT_INT4:
            printf('%s.%s has the value %d\n',
                rgprmbd[i].szPntName,
                rgprmbd[i].szPrmName,
                rgprmbd[i].pupvValue->int4);
            break;
        case DT_REAL:
            printf('%s.%s has the value %f\n',
                rgprmbd[i].szPntName,
                rgprmbd[i].szPrmName,
                rgprmbd[i].pupvValue->real);
            break;
        case DT_DOUBLE:
            printf('%s.%s has the value %f\n',
                rgprmbd[i].szPntName,
                rgprmbd[i].szPrmName,
                rgprmbd[i].pupvValue->double);
            break;
        case DT_ENUM:
            printf('%s.%s has the ordinal value %d
```

---

---

```

        and enum string %s\n',
            rgprmbd[0].szPntName,
            rgprmbd[i].szPrmName,
            rgprmbd[i].pupvValue->en.ord,
            rgprmbd[i].pupvValue->en.text);
        break;
    default:
        printf('Illegal type found\n');
        break;
    }
}
case NADS_PARTIAL_FUNC_FAIL:
    printf('rhsc_param_value_bynames partially failed');
    /* Check fStatus flags to find out which one(s)
       failed. */
    break;
default:
    printf('rhsc_param_value_bynames failed (c_geterrno()
           = 0x%x)', status);
break;
}

for (i=0; i<cprmbd; i++)
{
    free(rgprmbd[i].pupvValue);
}

```

---

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## See also

*rhsc\_param\_value\_puts\_2* on page 325

*rhsc\_param\_value\_put\_bynames* below

### **rhsc\_param\_value\_put\_bynames**

Control a list of point parameter values referenced by name.

### C/C++ Synopsis

```
int rhsc_param_put_bynames
(
    char*    szHostname,
    int      cprmbd,
    PARAM_BYNAME_DATA*
    rgprmbd
);
```

### VB Synopsis

```
RHSC_param_value_put_bynames (ByVal hostname As String,
    ByVal num_requests As Long,
    param_byname_data_array ()
    As param_byname_data) As Long
```

### Arguments

| Argument          | Description                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server that the data resides on                                                      |
| <b>cprmbd</b>     | (in) Number of controls to parameters values requested                                            |
| <b>rgprmbd</b>    | (in/out) Pointer to an array of PARAM_BYNAME_DATA structures (one array element for each request) |

### Description

The structure of the PARAM\_BYNAME\_DATA structure is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                            |                                   |
|----------------------------|-----------------------------------|
| <b>n_char* szPntName</b>   | (in) point name                   |
| <b>n_char* szPrmName</b>   | (in) parameter name               |
| <b>n_long nPrmOffset</b>   | (in) parameter offset             |
| <b>PARvalue* pupvValue</b> | (in) parameter value union        |
| <b>n_ushort nType</b>      | (in) value type                   |
| <b>n_long fStatus</b>      | (out) status of each value access |

**RHSC\_PARAM\_VALUE\_PUT\_BYNAMES** writes a list of point parameter values to the specified remote server and performs the necessary control. The control to point parameters are requested by name only and all name to number resolutions are performed by the server.

You can write a list of parameter values with different types using a single request. The value is placed into a union (of type **PARvalue**). Before storing the value to be written to a point parameter in the **PARAM\_VALUE\_DATA** structure, you must allocate sufficient memory for the union. You must free this memory before exiting your network application.

Although this is a list based function, there is no implication that it should be used as a sequential write function. If any individual put fails, the function will not prevent the remaining writes from occurring. The function will instead continue to write values to the remaining point parameters in the list.

Be careful when using **rhsc\_param\_value\_puts()** and **rhsc\_param\_value\_put\_bynames()** with more than one point/parameter pair. Each put causes a control to be executed on the server and each control takes a small amount of time. If more than one pair is put, the total time for each of these controls may exceed the default TCP/IP timeout. This will cause the Network API to report the error **RCV\_TIMEOUT**, even though all puts may have been successful. In addition, the Network API will be unavailable until the list of puts has been processed. This could cause subsequent calls to the network API to fail until the list is processed.

To simplify the handling of enumerations, two data types have been included for use with this function only. The data types are **DT\_ENUM\_ORD**, and **DT\_ENUM\_STR**. When writing a value to an enumeration point parameter, supply the ordinal part of the enumeration only and use the **DT\_ENUM\_ORD** data type. Alternatively, if you don't know the ordinal value, supply only the text component of the enumeration and use the **DT\_ENUM\_STR** data type. If the **DT\_ENUM** data type is specified, only the ordinal value is used by this function (similar to **DT\_ENUM\_ORD**).

A successful return status from the **rhsc\_param\_value\_put\_bynames** call indicates that no network errors were encountered (that is, the request was received, processed, and responded to).

If the returned value is **NADS\_PARTIAL\_FUNC\_FAIL**, then at least one request (and possibly all requests) failed. The status field of each array element should be checked to find which request failed. For each array element, a value of **CTLOK** (See *Diagnostics for Network API functions* on page 375) or 0 in the status field indicates that the control was successful.

The program using this function must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network.

Due to the ability to acquire a list of parameters of mixed data type, it is difficult to give generic limits. To meet the program requirement not to exceed the maximum packet size permitted, adhere to the following guidelines given for a number of specific cases:

- For request packets when writing **DT\_INT2** data only:

(10 \* number of points parameters) + sum of string lengths of point names  
+ sum of string lengths of parameter names < 4000

■ For request packets when writing DT\_INT4 data only:

(12 \* number of points parameters) + sum of string lengths of point names  
+ sum of string lengths of parameter names < 4000

■ For request packets when writing DT\_REAL data only:

(12 \* number of points parameters) + sum of string lengths of point names  
+ sum of string lengths of parameter names < 4000

■ For request packets when writing DT\_DBLE data only:

(16 \* number of points parameters) + sum of string lengths of point names  
+ sum of string lengths of parameter names < 4000

■ For request packets when writing DT\_CHAR data only:

(9 \* number of points parameters) + sum of string lengths of point names  
+ sum of string lengths of parameter names + sum of parameter value  
string lengths < 4000

■ For request packets when writing DT\_ENUM\_ORD data only:

(12 \* number of points parameters) + sum of string lengths of point names  
+ sum of string lengths of parameter names < 4000

■ For request packets when writing DT\_ENUM\_STR data only:

(9 \* number of points parameters) + sum of string lengths of point names  
+ sum of string lengths of parameter names + sum of parameter value enu-  
meration string lengths < 4000

■ For ALL reply packets:

(4 \* number of point parameters) < 4000

---

**Example**

Control the SP value of 'pntana1' to 42.0 and change its DESC to say  
'FunkyDescription.'

---

---

```
int status;
int i;

/* Set the point names, parameter names and parameter offsets to appropriate values */
PARAM_BYNAME_DATA rgprmbd[] = {{'pntanal','SP', 1},{'pntanal', 'DESC', 1}};
#define cprmbd (sizeof(rgprmbd)/sizeof(PARAM_BYNAME_DATA))

/* Allocate space and set the value and type to control pntanal.SP to 42.0 */
rgprmbd[0].pupvValue = (PARvalue *)malloc(sizeof(DT_REAL));
rgprmbd[0].pupvValue->real = (float)42.0;
rgprmbd[0].nType = DT_REAL;

/* Allocate space and set the value and type to control pntanal.DESC to 'FunkyDescription' */
rgprmbd[1].pupvValue = (PARvalue *)malloc(strlen('FunkyDescription')+1);
strcpy(rgprmbd[1].pupvValue->text, 'FunkyDescription');
rgprmbd[1].nType = DT_CHAR;

status = rhsc_param_value_put_bynames('server1', cprmbd, rgprmbd);

switch (status)
{
case 0:
    printf('rhsc_param_value_put_bynames successful\n');
    break;
case NADS_PARTIAL_FUNC_FAIL:
    printf('rhsc_param_value_put_bynames partially failed');
    /* Check fStatus flags to find out which one(s) failed. */
    break;
default:
    printf('rhsc_param_value_bynames failed (c_geterrno() = 0x%x)',
status);
    break;
}
```

---

---

```

}

for (i=0; i<cprmbd; i++)
{
    free (rgprmbd[i].pupvValue);
}

```

---

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## See also

*rhsc\_param\_values\_2* on page 330

*rhsc\_param\_value\_put\_bynames* on page 318

## rhsc\_param\_value\_put\_sec\_bynames

This function acts similarly to *rhsc\_param\_value\_put\_bynames* on page 318, except that it has an extra Station-related argument.

## C/C++ Synopsis

```

int rhsc_param_put_sec_bynames
(
    char*                szHostname,
    int                  cprmbd,
    PARAM_BYNAME_DATA*  rgprmbd,
    unsigned short       nStn
);

```

## VB Synopsis

```

RHSC_param_value_put_sec_bynames (ByVal hostname As String,
    ByVal num_requests As Long,
    param_byname_data_array() As param_byname_data,
    Station As short) As Long

```

## Arguments

| Argument          | Description                                                                                                                                                                                                                                                                                                                    |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server that the data resides on                                                                                                                                                                                                                                                                                   |
| <b>cprmbd</b>     | (in) Number of controls to parameters values requested                                                                                                                                                                                                                                                                         |
| <b>rgprmbd</b>    | (in/out) Pointer to an array of PARAM_BYNAME_DATA structures (one array element for each request)                                                                                                                                                                                                                              |
| <b>nStn</b>       | (in) Station number, which the Network Server uses in the associated CHANGE event for this call.<br><br>If operator-based security is used, the operator's name/ID will also be captured.<br><br>Note that even if the Station is not connected to the server, events raised by this function will still be logged against it. |

## Description

Unlike most Network API functions, events raised by this function are associated with the specified Station. (Events raised by other functions are associated with 'Network Server'.) If you want to control what events are logged by an application, see "Controlling what events are logged by an external application".

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## Controlling what events are logged by an external application

Bits 14 and 15 in **sysflg** (file 8, record 1, word 566) controls which events of an external application (such as Network API) are logged.

- Bit 14 determines whether only the two security functions (rhsc\_param\_value\_puts\_sec and rhsc\_param\_value\_put\_sec\_bynames) are logged.
- Bit 15 determines whether all events from an external application are logged or not.

For example, to only log events raised by rhsc\_param\_value\_puts\_sec or rhsc\_param\_value\_put\_sec\_bynames, set bit 14 to **On** and bit 15 to **Off**.

| Bit 14 | Bit 15 | Log all events from an external application | Log only events with security information |
|--------|--------|---------------------------------------------|-------------------------------------------|
| 0      | 0      | No                                          | No                                        |
| 1      | 0      | No                                          | Yes                                       |

| Bit 14 | Bit 15 | Log all events from an external application | Log only events with security information |
|--------|--------|---------------------------------------------|-------------------------------------------|
| 0      | 1      | Yes                                         | Yes                                       |
| 1      | 1      | Yes                                         | Yes                                       |

The events that are logged by an external application that uses Network Server or OPC Server are determined by two check boxes on the **Alarm/Event Options** tab of the Server Wide Settings display:

- Log Network Server and OPC Server changes to the database as events
- Log Network Server changes with security information to the database as events

There are three possible scenarios:

- If you do not want events logged, clear both check boxes
- If you only want events that have been raised via rhsc\_param\_value\_puts\_sec or rhsc\_param\_value\_put\_sec\_bynames, then:
  - Clear the **Log Network Server and OPC Server changes to the database as events** check box
  - Select the **Log Network Server changes with security information to the database as events** check box
- If you want all events logged, then select the **Log Network Server and OPC Server changes to the database as events** check box.

### rhsc\_param\_value\_puts\_2

Control a list of point parameter values.

### C Synopsis

```

int rhsc_param_value_puts_2
(
    char*    szHostname,
    int      cprmvd,
    PARAM_VALUE_DATA_2*  rgprmvd2
);

```

### VB Synopsis

```

rhsc_param_value_puts_2
(ByVal hostname As String,

```

```
ByVal num_requests As Long,
Param_value_data_array ()
As param_value_data_2) As Long
```

## Arguments

| Argument          | Description                                                                                      |
|-------------------|--------------------------------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server that the database resides on                                                 |
| <b>cprmvd</b>     | (in) The number of controls to parameters requested                                              |
| <b>rgprmvd2</b>   | (in/out) Pointer to a series of PARAM_VALUE_DATA_2 structures (one array element for each point) |

## Description

**rhsc\_param\_value\_puts\_2** is the replacement for the now deprecated **rhsc\_param\_value\_puts**.

---

### Attention:

**rhsc\_param\_value\_puts** can only access points in the range: 1 <= point number <= 65,000

---

The structure of the PARAM\_VALUE\_DATA\_2 structure is defined in **netapi\_types.h**. This structure and its members are defined as follows:

|                            |                              |
|----------------------------|------------------------------|
| <b>n_ulong nPnt</b>        | (in) point number            |
| <b>n_ushort nPrm</b>       | (in) parameter number        |
| <b>n_long nPrmOffset</b>   | (in) point parameter offset  |
| <b>PARvalue* pupvValue</b> | (in) parameter value union   |
| <b>n_ushort nType</b>      | (in) value type              |
| <b>n_long fStatus</b>      | (out) status of each request |

**RHSC\_PARM\_VALUE\_PUTS\_2** writes a list of point parameter values to the specified remote server and performs the necessary control. A function return of 0 is given if the point parameter values are successfully controlled, otherwise, an error code is returned.

You can write a list of parameter values with different types using a single request. The value is placed into a union (of type `PARvalue`). Before storing the value to be written to a point parameter in the `PARAM_VALUE_DATA_2` structure, you must allocate sufficient memory for the union. You must free this memory before exiting your network application.

Although this is a list based function, there is no implication that it should be used as a sequential write function. If any individual put fails, the function will not prevent the remaining writes from occurring. The function will instead continue to write values to the remaining point parameters in the list.

Be careful when using `rhsc_param_value_puts_2()` and `rhsc_param_value_put_bynames()` with more than one point/parameter pair. Each put causes a control to be executed on the server and each control takes a small amount of time. If more than one pair is put, the total time for each of these controls may exceed the default TCP/IP timeout. This will cause the Network API to report the error `RCV_TIMEOUT`, even though all puts may have been successful. In addition, the Network API will be unavailable until the list of puts has been processed. This could cause subsequent calls to the network API to fail until the list is processed.

To simplify the handling of enumerations, two data types have been included for use with this function only. The data types are `DT_ENUM_ORD`, and `DT_ENUM_STR`. When writing a value to an enumeration point parameter, supply the ordinal part of the enumeration only and use the `DT_ENUM_ORD` data type. Alternatively, if you don't know the ordinal value, supply only the text component of the enumeration and use the `DT_ENUM_STR` data type. If the `DT_ENUM` data type is specified, only the ordinal value is used by this function (similar to `DT_ENUM_ORD`).

A successful return status from the `rhsc_param_value_puts_2` call indicates that no network errors were encountered (that is, the request was received, processed, and responded to).

If the returned value is `NADS_PARTIAL_FUNC_FAIL`, then at least one request (and possibly all requests) failed. The status field of each array element should be checked to find which request failed. For each array element, a value of `CTLOK` (See Diagnostics for Network API functions) or 0 in the status field indicates that the control was successful.

The program using this function call must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network.

Due to the ability to control a list of parameters of mixed data type, it is difficult to give generic limits. To meet the program requirement not to exceed the maximum packet size permitted, adhere to the following guidelines given for a number of specific cases:

- For request packets when writing `DT_INT2` data only:

`(14 * number of points parameters) < 4000`

- For request packets when writing `DT_INT4` data only:

`(16 * number of points parameters) < 4000`

- For request packets when writing DT\_REAL data only:

(16 \* number of points parameters) < 4000

- For request packets when writing DT\_DBLE data only:

(20 \* number of points parameters) < 4000

- For request packets when writing DT\_CHAR data only:

(13 \* number of points parameters) + sum of string lengths of value character strings in bytes < 4000

- For request packets when writing DT\_ENUM\_ORD data only:

(13 \* number of points parameters) < 4000

- For request packets when writing DT\_ENUM\_STR data only:

(13 \* number of points parameters) + sum of string lengths of value enumeration strings in bytes < 4000

- For ALL reply packets:

(6 \* number of point parameters) < 4000

### Example

Control pntana1's SP value to 42.0 and change its DESC to say 'Funky description.'

```
int    status;
int    i;

POINT_NUMBER_DATA_2    rgpntnd2[] = {"pntana1"};
PARAM_NUMBER_DATA_2    rgprmnd2[] = {{0, "SP"},{0, "DESC"}};

#define cpntnd sizeof(rgpntnd2)/sizeof(POINT_NUMBER_DATA_2)
#define cprmnd sizeof(rgprmnd2)/sizeof(PARAM_NUMBER_DATA_2)
/* There are the same number of PARAM_VALUE_DATA entries as cprmnd. */
#define cprmvd sizeof(rgprmnd2)/sizeof(PARAM_NUMBER_DATA_2)

PARAM_VALUE_DATA_2    rgprmvd2[cprmvd];
```

---

```
status = rhsc_point_numbers_2("Server1", cpntnd, rgpntnd2);

rgprmnd2[0].nPnt = rgpntnd2[0].nPnt;
rgprmnd2[1].nPnt = rgpntnd2[0].nPnt;
status = rhsc_param_numbers_2("Server1", cprmnd, rgprmnd2);

/* Set the point number, parameter number and offset for the point
parameter. Allocate space, assign a value, and set the type for
pntana1.PV */
Rgprmvd2[0].nPnt = rgprmnd2[0].nPnt;
Rgprmvd2[0].nPrm = rgprmnd2[0].nPrm;
Rgprmvd2[0].nPrmoffset = 1 /* Set
parameter offset to default value*/
Rgprmvd2[0].pupvValue = (PARvalue *)malloc(sizeof(DT_REAL));
Rgprmvd2[0].pupvValue->real = (float)42.0;
Rgprmvd2[0].nType = DT_REAL;

/* Set the point number, parameter number and offset for the point
parameter. Allocate space, assign a value, and set the type for
pntana1.DESC */
Rgprmvd2[1].nPnt = rgprmnd2[1].nPnt;
Rgprmvd2[1].nPrm = rgprmnd2[1].nPrm;
Rgprmvd2[1].nPrmoffset = 1 /* Set
parameter offset to default value*/
Rgprmvd2[1].pupvValue =
    (PARvalue *)malloc(strlen("Funky description") + 1);
strcpy(rgprmvd2[1].pupvValue->text, "Funky description");
Rgprmvd2[1].nType = DT_CHAR;

status = rhsc_param_value_puts_2("Server1", cprmvd, rgprmvd2);
switch (status)
{
    case 0:
        printf("rhsc_param_value_puts_2 successful\n");
        break;
    case NADS_PARTIAL_FUNC_FAIL:
        printf("rhsc_param_value_puts_2 partially failed\n");
}
```

---

---

```

        /* Check fStatus flags to find out which ones failed. */
        break;
    default:
        printf("rhsc_param_value_puts_2 failed(c_geterrno() =
0x%x)\n", status);
        break;
}

for (i=0; i<cprmvd; i++)
{
    free(rgprmvd2[i].pupvValue);
}

```

---

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## See also

*rhsc\_param\_values\_2* below

*rhsc\_param\_value\_put\_bynames* on page 318

## rhsc\_param\_values\_2

Read a list of point parameter values.

## C Synopsis

```

int rhsc_param_values_2
(
    char*          szHostname,
    int           nPeriod,
    int           cprmvd,
    PARAM_VALUE_DATA_2* rgprmvd2
);

```

## VB Synopsis

```

rhsc_param_values_2 (ByVal hostname As String,
    ByVal period as Long,
    ByVal num_requests as Long,

```

```
param_value_data_array()  
As param_value_data_2) As Long
```

## Arguments

| Argument          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server that the database resides on.                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>nPeriod</b>    | (in) subscription period in milliseconds for the point parameters. Use the constant NADS_READ_CACHE if subscription is not required. If the value is in the Experion cache, then that value will be returned. Otherwise the controller will be polled for the latest value. Use the constant NADS_READ_DEVICE if you want to force Experion to re-poll the controller. The subscription period will not be applied to standard point types. |
| <b>cprmvd</b>     | (in) The number of parameter values requested.                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>rgprmvd2</b>   | (in/out) Pointer to an array of PARAM_VALUE_DATA_2 structures (one array element for each request).                                                                                                                                                                                                                                                                                                                                         |

## Description

rhsc\_param\_values\_2 is the replacement for the now deprecated rhsc\_param\_values.

---

### Attention:

rhsc\_param\_values can only access points in the range: 1 <= point number <= 65,000.

---

The structure of the PARAM\_VALUE\_DATA\_2 structure is defined in **netapi\_types.h**. This structure and its members are defined as follows:

|                            |                              |
|----------------------------|------------------------------|
| <b>n_ulong nPnt</b>        | (in) point number            |
| <b>n_ushort nPrm</b>       | (in) parameter number        |
| <b>n_long nPrmOffset</b>   | (in) point parameter offset  |
| <b>PARvalue* pupvValue</b> | (out) parameter value union  |
| <b>n_ushort nType</b>      | (out) value type             |
| <b>n_long fStatus</b>      | (out) status of each request |

If your system uses *dynamic scanning*, rhsc\_param\_values\_2 calls from the Network API do not trigger dynamic scanning.

`RHSC_PARM_VALUES_2` requests a list of point parameter values from the specified remote server. A function return of 0 is given if the parameter values were successfully read else an error code is returned.

You can read a list of parameter values with different types using a single request. Each point parameter value is placed into a union (of type `PARvalue`). Before making the request, you must allocate sufficient memory for each value union. You must free this memory before exiting your Network application.

A successful return status from the `rhsc_param_values_2` call indicates that no network errors were encountered (that is, the request was received, processed, and responded to). If the returned value is `NADS_PARTIAL_FUNC_FAIL`, then at least one request (and possibly all requests) failed. The status field of each array element should be checked to find which request failed.

The program using this function call must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network.

Due to the ability to acquire a list of parameters of mixed data type, it is difficult to give generic limits. To meet the program requirement not to exceed the maximum packet size permitted, adhere to the following guidelines given for a number of specific cases:

- For ALL request packets:

$(10 * \text{number of point parameters}) < 4000$

- For response packets when reading `DT_INT2` data only:

$(10 * \text{number of points parameters}) < 4000$

- For response packets when reading `DT_INT4` data only:

$(12 * \text{number of points parameters}) < 4000$

- For response packets when reading `DT_REAL` data only:

$(12 * \text{number of points parameters}) < 4000$

- For response packets when reading `DT_DBLE` data only:

$(16 * \text{number of points parameters}) < 4000$

- For response packets when reading `DT_CHAR` data only:

$(9 * \text{number of points parameters}) + \text{sum of string lengths of value character strings in bytes} < 4000$

- For response packets when reading `DT_ENUM` data only:

$(13 * \text{number of points parameters}) + \text{sum of string lengths of value enumeration strings in bytes} < 4000$

---

**Example****Read the value of pntana1.SP and pntana1.DESC.**

```

int    status;
int    i;
POINT_NUMBER_DATA_2  rgpntnd2[] = {{'pntana1'}};
PARAM_NUMBER_DATA_2  rgprmnd2[] = {{0, 'SP'}, {0, 'DESC'}};

#define cpntnd sizeof(rgpntnd2)/sizeof(POINT_NUMBER_DATA_2)
#define cprmnd sizeof(rgprmnd2)/sizeof(PARAM_NUMBER_DATA_2)
/* There are the same number of PARAM_VALUE_DATA_2 entries as cprmnd.
*/
#define cprmvd sizeof(rgprmnd2)/sizeof(PARAM_NUMBER_DATA_2)

PARAM_VALUE_DATA_2  rgprmvd2[cprmvd];

status = rhsc_point_numbers_2("server1", cpntnd, rgpntnd2);
rgprmnd2[0].nPnt = rgpntnd2[0].nPnt;
rgprmnd2[1].nPnt = rgpntnd2[0].nPnt;
status = rhsc_param_numbers_2("server1", cprmnd, rgprmnd2);

for (i=0; i<cprmvd; i++)
{
    rgprmvd2[i].nPnt = rgprmnd2[i].nPnt;
    rgprmvd2[i].nPrm = rgprmnd2[i].nPrm;
    /*Use of the parameter offset is currently unsupported.
    Set offset to the default value 1. */
    rgprmvd2[i].nPrmOffset = 1;
}

/*
ALLOCATING MEMORY:
Sufficient memory must be allocated for each value union. If the
value type is not known, allocate memory for the largest possible
size of a PARvalue union. See below for an example of how to allocate
this memory.
If the data type is known, then allocate the exact amount of memory

```

---

---

```
to save space.
For example for DT_REAL values:
    rgprmvd2[0].pupvValue = (PARvalue *) malloc(sizeof(DT_REAL));
*/
for (i=0; i<cprmvd; i++)
{
    rgprmvd2[i].pupvValue = (PARvalue *)malloc(sizeof(PARvalue));

status = rhsc_param_values_2('server1',
NADS_READ_CACHE, cprmvd, rgprmvd2);

switch (status)
{
    case 0:
        printf('rhsc_param_values_2 successful\n');
        for (i=0; i<cprmvd; i++)
        {
            switch (rgprmvd2[i].nType)
            {
                case DT_CHAR:
                    printf('%s.%s has the value %s\n',
                        rgpntnd2[0].szPntName,
                        rgprmnd2[i].szPrmName,
                        rgprmvd2[i].pupvValue->text);
                    break;
                case DT_INT2:
                    printf('%s.%s has the value %d\n',
                        rgpntnd2[0].szPntName,
                        rgprmnd2[i].szPrmName,
                        rgprmvd2[i].pupvValue->int2);
                    break;
                case DT_INT4:
                    printf('%s.%s has the value %d\n',
                        rgpntnd2[0].szPntName,
                        rgprmnd2[i].szPrmName,
                        rgprmvd2[i].pupvValue->int4);
                    break;
            }
        }
    }
}
```

---

---

```
        case DT_REAL:
            printf('%s.%s has the value %f\n',
                rgpntnd2[0].szPntName,
                rgprmnd2[i].szPrmName,
                rgprmvd2[i].pupvValue->real);
            break;
        case DT_DBLE:
            printf('%s.%s has the value %f\n',
                rgpntnd2[0].szPntName,
                rgprmnd2[i].szPrmName,
                rgprmvd2[i].pupvValue->dbble);
            break;
        case DT_ENUM:
            printf('%s.%s has the ordinal value
                %d and enum string %s\n',
                rgpntnd2[0].szPntName,
                rgprmnd2[i].szPrmName,
                rgprmvd2[i].pupvValue->en.ord,
                rgprmvd2[i].pupvValue->en.text);
            break;
        default:
            printf('Illegal type found\n');
            break;
    }
}
break;
case NADS_PARTIAL_FUNC_FAIL:
    printf('rhsc_param_values_2 partially failed\n');
    /* Check fStatus flags to find out which ones failed. */
    break;
default:
    printf('rhsc_param_values_2 failed (c_geterrno() = 0x%x)\n',
status);
    break;
}

for (i=0; i<cprmvd; i++)
```

---

```
{
    free (rgprmvd2 [i] .pupvValue);
}
```

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## See also

*rhsc\_param\_value\_puts\_2* on page 325

*rhsc\_param\_value\_put\_bynames* on page 318

## rhsc\_point\_numbers\_2

Resolve a list of point names to numbers.

## C/C++ Synopsis

```
int rhsc_point_numbers_2
(
    char*          szHostname,
    int            cpntnd,
    POINT_NUMBER_DATA_2*  rgpntnd2
);
```

## VB Synopsis

```
rhsc_point_numbers_2 (ByVal hostname As String,
    ByVal num_requests As Long,
    point_number_data_array()
    As Point_Number_Data_2) As Long
```

## Arguments

| Argument          | Description                                                               |
|-------------------|---------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server that the database resides on                          |
| <b>cpntnd</b>     | (in) The number of point name resolutions requested                       |
| <b>rgpntnd2</b>   | (in/out) Pointer to a series of POINT_NUMBER_DATA_2 structures (one array |

| Argument | Description               |
|----------|---------------------------|
|          | element for each request) |

## Description

**rhsc\_point\_numbers\_2** is the replacement for the now deprecated **rhsc\_point\_numbers**.

---

### Attention:

rhsc\_point\_numbers can only access points in the range: 1 <= point number <= 65,000.

---

The structure of the POINT\_NUMBER\_DATA\_2 structure is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                          |                              |
|--------------------------|------------------------------|
| <b>n_char* szPntName</b> | (in) point name to resolve   |
| <b>n_ulong nPnt</b>      | (out) point number           |
| <b>n_long fStatus</b>    | (out) status of each request |

RHSC\_POINT\_NUMBERS\_2 converts a list of point names to their equivalent point numbers for a specified remote server.

A successful return status from the **rhsc\_point\_numbers\_2** call indicates that no network errors were encountered (that is, the request was received, processed, and responded to). If the returned value is NADS\_PARTIAL\_FUNC\_FAIL, then at least one request (and possibly all requests) failed. The status field of each array element should be checked to find which request failed.

The program using this function call must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network. To meet this program requirement, adhere to the following guidelines:

- For request packets:

```
(4 * number of points) + sum of string lengths of point names in bytes < 4000
```

- For response packets:

```
(8 * number of points) < 4000
```

## Example

Resolve the point names 'pntana1' and 'pntana2.'

```
int    status;
int    i;
POINT_NUMBER_DATA_2  rgpntnd2[] = {{'pntana1'}, {'pntana2'}};
#define cpntnd sizeof(rgpntnd2)/sizeof(POINT_NUMBER_DATA_2)

status = rhsc_point_numbers_2('Server1', cpntnd, rgpntnd2);

switch (status)
{
    case 0:
        printf('rhsc_point_numbers_2 successful\n');
        for (i=0; i<cpntnd; i++)
        {
            printf('%s has the point number %d\n',
                rgpntnd2[i].szPntName,
                rgpntnd2[i].nPnt);
        }
        break;
    case NADS_PARTIAL_FUNC_FAIL:
        printf('rhsc_point_numbers_2 partially failed\n');
        /* Check fStatus flags to find out which ones failed. */
        break;
    default:
        printf('rhsc_point_numbers_2 failed (c_geterrno() = 0x%x)\n',
            status);
        break;
}
```

---

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

### rputdat

Store a list of fields to a user file.

## C Synopsis

```
int rputdat(char *server,
            int num_points,
            rgetdat_data getdat_data[])
```

## VB Synopsis

```
rputdat_int(ByVal server As String,
            ByVal num_points As Integer,
            getdat_int_data() As rgetdat_int_data_str) As Integer
rputdat_bits(ByVal server As String,
            ByVal num_points As Integer,
            putdat_bits_data() As rgetdat_bits_data_str) As Integer
rputdat_long(ByVal server As String,
            ByVal num_points As Integer,
            getdat_long_data() As rgetdat_long_data_str) As Integer
rputdat_float(ByVal server As String,
            ByVal num_points As Integer,
            getdat_float_data()
            As rgetdat_float_data_str) As Integer
rputdat_double(ByVal server As String,
            ByVal num_points As Integer,
            getdat_double_data()
            As rgetdat_double_data_str) As Integer
rputdat_str(ByVal server As String,
            ByVal num_points As Integer,
            getdat_str_data()
            As rgetdat_str_data_str) As Integer
```

## Arguments

| Argument                | Description                                                                       |
|-------------------------|-----------------------------------------------------------------------------------|
| <b>server</b>           | (in) Name of server that the database resides on                                  |
| <b>num_points</b>       | (in) The number of points passed to rgetdat_xxxx in the getdat_xxxx_data argument |
| <b>getdat_xxxx_data</b> | (in/out) Pointer to a series of rgetdat_xxxx_data structures (one for each point) |

## Description

This function call enables fields from a user table to be changed. The fields to be accessed are referenced by the members of the `rgetdat_data` structure (see below). The function accepts an array of `rgetdat_data` structures thus providing the flexibility to set multiple fields with one call. Note that the fields can be of different types and from different database files.

A successful return status from the **rputdat** call indicates that no network error were encountered (that is, the request was received, processed and responded to). The status field in each call structure must still be checked on return to determine the result of the individual remote calls.

The structure of the `rgetdat_data` structure is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                       |                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>int2 type</b>      | (in) Defines the type of data to be retrieved/stored, this will be one of the standard server data types. Namely using one of the following defines:<br>RGETDAT_TYPE_INT2, RGETDAT_TYPE_INT4, RGETDAT_TYPE_REAL4, RGETDAT_TYPE_REAL8, RGETDAT_TYPE_STR, RGETDAT_TYPE_BITS                                                                                                                      |
| <b>int2 file</b>      | (in) Absolute database file number to retrieve/store field.                                                                                                                                                                                                                                                                                                                                    |
| <b>int2 rec</b>       | (in) Record number in above file to retrieve/store field.                                                                                                                                                                                                                                                                                                                                      |
| <b>int2 word</b>      | (in) Word offset in above record to retrieve/store field.                                                                                                                                                                                                                                                                                                                                      |
| <b>int2 start_bit</b> | (in) Start bit offset into the above word for the first bit of a bit sequence to be retrieved/stored. (that is, The bit sequence starts at: word + start_bit, start_bit=0 is the first bit in the field.). Ignored if type not a bit sequence.                                                                                                                                                 |
| <b>int2 length</b>    | (in) Length of bit sequence or string to retrieve/store, in characters for a string, in bits for a bit sequence. Ignored if type not a string or bit sequence.                                                                                                                                                                                                                                 |
| <b>int2 flags</b>     | (in) Bit zero specifies the direction to read/write for circular files. (0 = newest record, 1 = oldest record)                                                                                                                                                                                                                                                                                 |
| <b>union value</b>    | (in/out) Value of field retrieved or value to be stored. When storing strings they must be of the length given above. When strings are retrieved they become NULL terminated, hence the length allocated to receive the string must be one more than the length specified above. Bit sequences will start at bit zero and be length bits long. See below for a description of the union types. |
| <b>int2 status</b>    | (out) Return value of actual remote putdat/putdat call.                                                                                                                                                                                                                                                                                                                                        |

The union structure of the value member used in the `rgetdat_data` structure is defined in `nif_types.h`. This structure and its members is defined as follows:

|                            |                                                                                                                                                                                          |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>short int2</b>          | Two byte signed integer.                                                                                                                                                                 |
| <b>long int4</b>           | Four byte signed integer.                                                                                                                                                                |
| <b>float real4</b>         | Four byte IEEE floating point number.                                                                                                                                                    |
| <b>double real8</b>        | Eight byte (double precision) IEEE floating point number.                                                                                                                                |
| <b>char* str</b>           | Pointer to string to be stored. Note this string need not be NULL terminated, but must be of the length specified by length (see <code>rgetdat_data</code> structure description above). |
| <b>unsigned short bits</b> | Two byte unsigned integer to be used to for bit sequences (partial integer). (Note the maximum length of a bit sequence is limited to 16.)                                               |

The program using this function must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network. To meet this requirement, adhere to the following guideline:

```
(22 * number of fields) + sum of all string value lengths in bytes <4000
```

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## Backward-compatibility Functions

The following functions are available for backwards compatibility.

### `hsc_napierrstr`

Lookup an error string from an error number. This function is provided for backward compatibility.

### C/C++ Synopsis

```
void hsc_napierrstr(UINT err, LPSTR texterr);
```

### VB Synopsis

```
hsc_napierrstr(ByVal err As Integer) As String
```

## Arguments

| Argument       | Description                     |
|----------------|---------------------------------|
| <b>err</b>     | (in) The error number to lookup |
| <b>texterr</b> | (out) The error string returned |

## Diagnostics

This function will always return a usable string value.

### rgethstpar\_date

Retrieve history values for a Point based on date.

This function's synopsis and description are identical to that of 'rgethstpar\_ofst.'

### rgethstpar\_ofst

Retrieve history values for a point based on offset.

## C synopsis

```

    int rgethstpar_date
(
    char*                server,
    int                  num_gethsts,
    rgethstpar_date_data* gethstpar_date_data
);
int rgethstpar_ofst
(
    char*                server,
    int                  num_gethsts,
    rgethstpar_ofst_data* gethstpar_ofst_data
);

```

## VB synopsis

```

rgethstpar_date(ByVal server As String,
    gethstpar_date_data
    As rgethstpar_date_data_str) As Integer
rgethstpar_ofst(ByVal server As String,
    gethstpar_ofst_data
    As rgethstpar_ofst_data_str) As Integer

```

## Arguments

| Argument                   | Description                                                                             |
|----------------------------|-----------------------------------------------------------------------------------------|
| <b>server</b>              | (in) Name of server that the database resides on                                        |
| <b>num_gethsts</b>         | (in) The number of points passed to rgethstpar_xxxx in the gethstpar_xxxx_data argument |
| <b>gethstpar_xxxx_data</b> | (in/out) Pointer to a series of rgethstpar_xxxx_data structures (one for each point)    |

## Description

This function is provided for backwards compatibility. These functions can only access points in the range: 1 <= point number <=65,000.

The functions rhsc\_param\_hist\_dates\_2 and rhsc\_param\_hist\_offsets\_2 should be used instead.

Use this function to retrieve history values for points. The two types of history (based on time or offset) are retrieved using the corresponding function variation. History will be retrieved from a specified time or offset going backwards in time. The history values to be accessed are referenced by the rgethst\_date\_data and rgethst\_ofst\_data structures (see below). The functions accept an array of these structures, thus providing access to multiple point history values with one function call.

Note that a successful return status from the rgethst call indicates that no network errors were encountered (that is, the request was received, processed and responded to). The status field in each call structure needs to be verified on return to determine the result of the individual remote calls.

The structure of the rgethst\_date\_data structure is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                               |                                                                                                                                                                                                                                                     |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>uint2 hist_type</b>        | (in) Defines the type of history to retrieve, this will be one of the standard server history types. Namely using one of the following:<br><br>HST_1MIN, HST_6MIN, HST_1HOUR, HST_8HOUR, HST_24HOUR, HST_5SECF, HST_1HOURE, HST_8HOURE, HST_24HOURE |
| <b>uint4 hist_start_date</b>  | (in) Start date of history to receive in Julian days (number of days since 1st January 1981).                                                                                                                                                       |
| <b>ureal4 hist_start_time</b> | (in) Start time of history to retrieve in seconds since midnight.                                                                                                                                                                                   |
| <b>uint2 num_hist</b>         | (in) Number of history values per point to be retrieved.                                                                                                                                                                                            |

|                                 |                                                                                                                       |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>uint2 num_ points</b>        | (in) Number of points to be processed. MAXIMUM value allowed is 20.                                                   |
| <b>uint2* point_ type_ nums</b> | (in) Array (of dimension num_ points) containing the point type/numbers of the point history values to retrieve.      |
| <b>uint2* point_ params</b>     | (in) Array of (dimension num_ points) containing the parameter numbers of the history values to retrieve.             |
| <b>uchar* archive_ path</b>     | This member is no longer in use and is only retained for backwards compatibility. Instead, pass a zero length string. |
| <b>real4* hist_ values</b>      | (out) Array (of dimension num_ points * num_ hist) to provide storage for the returned history values.                |
| <b>uint2 gethst_ status</b>     | (out) Return value of the actual remote gethst_ date call.                                                            |

The structure of the rgethst\_ofst\_data structure is defined in nif\_types.h. This structure and its members are defined as follows:

|                                 |                                                                                                                                                                                                                                                      |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>uint2 hist_ type</b>         | (in) Defines the type of history to retrieve, this will be one of the standard server history types. Namely using one of the following defines: HST_1MIN, HST_6MIN, HST_1HOUR, HST_8HOUR, HST_24HOUR, HST_5SECF, HST_1HOURE, HST_8HOURE, HST_24HOURE |
| <b>uint4 hist_ offset</b>       | (in) Offset from latest history value in history intervals where offset=1 is the most recent history value).                                                                                                                                         |
| <b>uint2 num_ hist</b>          | (in) Number of history values per point to be retrieved.                                                                                                                                                                                             |
| <b>uint2 num_ points</b>        | (in) Number of points to be processed. MAXIMUM value allowed is 20.                                                                                                                                                                                  |
| <b>uint2* point_ type_ nums</b> | (in) Array (of dimension num_ points) containing the point type/numbers of the point history values to retrieve.                                                                                                                                     |
| <b>uint2* point_ params</b>     | (in) Array of (dimension num_ points) containing the parameter numbers of the history values to retrieve.                                                                                                                                            |
| <b>uchar* archive_ path</b>     | This member is no longer in use and is only retained for backwards compatibility. Instead, pass a zero length string.                                                                                                                                |

|                                     |                                                                                                      |
|-------------------------------------|------------------------------------------------------------------------------------------------------|
| <b>real4*<br/>hist_<br/>values</b>  | (out) Array (of dimension num_points * num_hist) to provide storage for the returned history values. |
| <b>uint2<br/>gethst_<br/>status</b> | (out) Return value of the actual remote gethst_ofst call.                                            |

The program using this function call must ensure that the size of the network packets generated does not exceed the maximum packet size permitted on the network. To meet this requirement, adhere to the following guideline:

- For request packets for rgethst\_date:

```
(15 * number of history requests) + (2 * number of points requested) +
string lengths of archive paths
< 4000.
```

- For request packets for rgethist\_ofst:

```
(11 * number of history requests) + (2 * number of points requested) +
string lengths of archive paths
<4000.
```

- For response packets:

```
(4 * number of history requests) + (4 * (For each history request the sum
of (num_hist * num_points)))
```

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

### rgetpnt

Get point type/number by point name string.

## C Synopsis

```
int rgetpnt
(
    char*          server,
    int            num_points,
    rgetpnt_data* getpnt_data
);
```

## VB Synopsis

```
rgetpnt (ByVal server As String,
        ByVal num_points As Integer,
        getpnt_data() As rgetpnt_data_str) As Integer
```

## Arguments

| Argument           | Description                                                                  |
|--------------------|------------------------------------------------------------------------------|
| <b>server</b>      | (in) Name of server that the database resides on                             |
| <b>num_points</b>  | (in) The number of points passed to rgetpnt in the getpnt_data argument      |
| <b>getpnt_data</b> | (in/out) Pointer to a series of rgetpnt_data structures (one for each point) |

## Description

This function is provided for backwards compatibility. It cannot be used to access point information for points on Process Controllers. This function can only access points in the range: 1 <= point number <=65,000.

The rhsc\_point\_numbers\_2 function should be used instead.

This function enables the point type/number to be resolved from the point name. Each point name to be resolved is stored in a rgetpnt\_data structure. The function accepts an array of structures, thus enabling multiple point names to be resolved with one function call.

The structure of rgetpnt\_data is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                             |                                                                                                        |
|-----------------------------|--------------------------------------------------------------------------------------------------------|
| <b>char* point_name</b>     | (in) Pointer to a null terminated string containing the point name to be resolved into a point number. |
| <b>uint2 point_type_num</b> | (out) Return value of the point type/number for the point named above.                                 |
| <b>uint2 getpnt_status</b>  | (out) Return value of the actual remote getpnt call.                                                   |

Note that a successful return status from the **rgetpnt** call indicates that no network errors, were encountered (that is, the request was received, processed and responded to). The status field in each call structure needs to be verified on return to determine the result of the individual remote calls.

The program using this function call must ensure that the size of the network packets generated does not exceed the maximum packet size permitted on the network. To meet this requirement, adhere to the following guideline:

(4 \* number of points) + sum of string lengths of point names in bytes  
<4000

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## See also

*rhsc\_point\_numbers\_2* on page 336

### rgetval\_num

Retrieve the value of a numeric point parameter.

This function's synopsis and description are identical to that of 'rgetval\_hist.'

### rgetval\_ascii

Retrieve the value of an ASCII point parameter.

This function's synopsis and description are identical to that of 'rgetval\_hist.'

### rgetval\_hist

Retrieve the value of a history point parameter.

## C Synopsis

```
int rgetval_num
(
    char*          server,
    int            num_points,
    rgetval_num_data*  getval_num_data
);
int rgetval_ascii
(
    char*          server,
    int            num_points,
    rgetval_ascii_data*  getval_ascii_data
);
int rgetval_hist
(
    char*          server,
    int            num_points,
```

```

        rgetval_hist_data*  getval_hist_data
    );

```

### VB Synopsis

```

rgetval_numb(ByVal server As String,
             ByVal num_points As Integer,
             getval_num_data() As rgetval_num_data_str)
             As Integer
rgetval_ascii(ByVal server As String,
              ByVal num_points As Integer,
              getval_ascii_data() As rgetval_ascii_data_str)
              As Integer
rgetval_hist(ByVal server As String,
             ByVal num_points As Integer,
             getval_hist_data() As rgetval_hist_data_str)
             As Integer

```

### Arguments

| Argument                | Description                                                                       |
|-------------------------|-----------------------------------------------------------------------------------|
| <b>server</b>           | (in) Name of server that the database resides on                                  |
| <b>num_points</b>       | (in) The number of points passed to rgetval_xxxx in the getval_xxxx_data argument |
| <b>getval_xxxx_data</b> | (in/out) Pointer to a series of rgetval_xxxx_data structures (one for each point) |

### Description

This function is provided for backwards compatibility. It cannot be used to access point information for points on Process Controllers. This function can only access points in the range: 1 <= point number <=65,000.

The rhsc\_param\_values\_2 function should be used instead.

This function call enables access to point parameter values. The three types of parameters (numerical, ASCII and history) are accessed using the corresponding function variations. The point parameters to be accessed are referenced in the rgetval\_num\_data, rgetval\_ascii\_data and rgetval\_hist\_data structures (see below). The functions accept an array of structures, thus providing access to multiple point parameter values with one call.

The structure of rgetval\_num\_data is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                                           |                                                                                                                                                                                                         |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>uint2<br/>point_<br/>type_<br/>num</b> | (in) Defines the point type/number to be accessed.                                                                                                                                                      |
| <b>uint2<br/>point_<br/>param</b>         | (in) Defines the point parameter to be accessed. (for example, process variable (PV), Mode (MD), Output (OP) or Set Point (SP)). The definitions for all parameters are located in the parameters file. |
| <b>real4<br/>param_<br/>value</b>         | (out) Value of the point parameter retrieved.                                                                                                                                                           |
| <b>uint2<br/>getval_<br/>status</b>       | (out) The return value of the actual remote getval call.                                                                                                                                                |

The structure of `rgetval_ascii_data` is defined in `nif_types.h`. This structure and its members are defined as follows:

|                                           |                                                                                                                                                                                                                                               |
|-------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>uint2<br/>point_<br/>type_<br/>num</b> | (in) Defines the point type/number to be accessed.                                                                                                                                                                                            |
| <b>uint2<br/>point_<br/>param</b>         | (in) Defines the point parameter to be accessed (for example, description (DESC)). The definitions for all parameter types are located in the parameters file.                                                                                |
| <b>char*<br/>param_<br/>value</b>         | (out) NULL terminated string value of the point parameter. Note that this string can have a length of <code>NIF_MAX_ASCII_PARAM_LEN + 1</code> (for the termination), and that this amount of space must be allocated by the calling program. |
| <b>uint2<br/>param_<br/>len</b>           | (out) Useful length of above <code>param_value</code> retrieved (in bytes).                                                                                                                                                                   |
| <b>uint2<br/>getval_<br/>status</b>       | (out) The return value of the actual remote getval call.                                                                                                                                                                                      |

The structure of the `rgetval_hist_data` structure is defined in `nif_types.h`. This structure and its members are defined as follows:

|                                           |                                                                                                                                                                                 |
|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>uint2<br/>point_<br/>type_<br/>num</b> | (in) Defines the point type/number to be accessed.                                                                                                                              |
| <b>uint2<br/>point_<br/>param</b>         | (in) Defines the point parameter to be accessed (for example, 1 minute history, <code>HST_1MIN</code> ). The definitions for all parameters are located in the parameters file. |

|                            |                                                                                                                                     |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>param</b>               |                                                                                                                                     |
| <b>uint2 hist_offset</b>   | (in) Offset from latest history value in history intervals to retrieve value, where hist_offset=1 is the most recent history value. |
| <b>real4 param_value</b>   | (out) Value of the point parameter retrieved.                                                                                       |
| <b>uint2 getval_status</b> | (out) The return value of the actual remote getval call.                                                                            |

Note that a successful return status from the **rgetval** call indicates that no network errors were encountered (that is, the request was received, processed and responded to). The status field in each call structure must still be checked on return to determine the result of the individual remote calls.

The program using these function calls must ensure that the size of the network packets generated does not exceed the maximum packet size permitted on the network. This requirement can be met by adhering to the following guideline:

$$(12 * \text{number of points}) + \text{sum of all string value lengths in bytes} < 4000$$

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## See also

*rhsc\_param\_values\_2* on page 330

*rhsc\_param\_value\_puts\_2* on page 325

## rgetpntval

Get the numeric parameter value.

This function's synopsis and description are identical to that of 'rgetpntval\_ascii.'

## rgetpntval\_ascii

Get the ASCII parameter value.

## VB Synopsis

```
rgetpntval (ByVal server As String,
            ByVal point As String,
            ByVal param As Integer,
```

```

        value As Single) As Integer
rgetpntval_ascii (ByVal server As String,
        ByVal point As String,
        ByVal param As Integer,
        value As String,
        ByVal length As Integer) As Integer
    
```

### Arguments

| Argument      | Description                                                    |
|---------------|----------------------------------------------------------------|
| <b>server</b> | (in) Name of server that the database resides on               |
| <b>point</b>  | (in) Name of point                                             |
| <b>param</b>  | (in) point parameter number                                    |
| <b>value</b>  | (out) Value of point parameter returned by function            |
| <b>length</b> | (in) Maximum length of the string returned by rgetpntval_ascii |

### Description

This function is provided for backwards compatibility. It cannot be used to access point information for points on Process Controllers. This function can only access points in the range: 1 <= point number <=65,000.

The **rhsc\_param\_values\_2** function should be used instead.

RGETPNTVAL and RGETPNTVAL\_ASCII provide VB interfaces to request a single parameter value that has the data types Single and String respectively. These functions can only be used to read one parameter value at a time. A function return of 0 is given if the parameter value was successfully read; else an error code is returned.

### Diagnostics

See *Diagnostics for Network API functions* on page 375.

#### rhsc\_param\_hist\_dates

---

**Attention:**

---

---

**rhsc\_param\_hist\_dates** is deprecated and may be removed in a future release. It is provided compatibility purposes only. When used with an Experion server release R400 or later **rhsc\_param\_hist\_dates** will only be able to access points in the range: 1 <= point number <= 65,000. The replacement function **rhsc\_param\_hist\_dates\_2** should be used instead.

---

Retrieve history values for a point based on date.

This function's synopsis and description are identical to that of 'rhsc\_param\_hist\_offsets.'

### rhsc\_param\_hist\_offsets

---

**Attention:**

**rhsc\_param\_hist\_offsets** is deprecated and may be removed in a future release. It is provided compatibility purposes only. When used with an Experion server release R400 or later **rhsc\_param\_hist\_offsets** will only be able to access points in the range: 1 <= point number <= 65,000. The replacement function **rhsc\_param\_hist\_offsets\_2** should be used instead.

---

Retrieve history values for a point based on offset.

### C/C++ Synopsis

```
int rhsc_param_hist_dates
(
    char*          server,
    int            num_gethsts,
    rgethstpar_date_data* gethstpar_date_data
);
int rhsc_param_hist_offsets
(
    char*          server,
    int            num_gethsts,
    rgethstpar_ofst_data* gethstpar_ofst_data
);
```

### VB Synopsis

```
rhsc_param_hist_dates(ByVal server As String,
    num_requests As Long,
```

```

    gethstpar_date_data_array()
    As gethstpar_date_data) As Long
rhsc_param_hist_offsets(ByVal server As String,
    num_requests As Long,
    gethstpar_ofst_data_array()
    As gethstpar_ofst_data) As Long

```

## Arguments

| Argument                   | Description                                                                                          |
|----------------------------|------------------------------------------------------------------------------------------------------|
| <b>server</b>              | (in) Name of server that the database resides on                                                     |
| <b>num_requests</b>        | (in) The number of history requests                                                                  |
| <b>gethstpar_xxxx_data</b> | (in/out) Pointer to an array of rgethstpar_xxxx_data structures (one array element for each request) |

## Description

Use this function to retrieve history values for points. The two types of history (based on time or offset) are retrieved using the corresponding function variation. History will be retrieved from a specified time or offset going backwards in time. The history values to be accessed are referenced by the rgethst\_date\_data and rgethst\_ofst\_data structures (see below). The functions accept an array of these structures, thus providing access to multiple point history values with one function call.

Note that a successful return status from the rgethst call indicates that no network errors were encountered (that is, the request was received, processed and responded to). The status field in each call structure needs to be verified on return to determine the result of the individual remote calls.

The structure of the rgethst\_date\_data structure is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                               |                                                                                                                                                                                                                                                 |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>uint2 hist_type</b>        | (in) Defines the type of history to retrieve, this will be one of the standard server history types. Namely using one of the following:<br>HST_1MIN, HST_6MIN, HST_1HOUR, HST_8HOUR, HST_24HOUR, HST_5SECF, HST_1HOURE, HST_8HOURE, HST_24HOURE |
| <b>uint4 hist_start_date</b>  | (in) Start date of history to receive in Julian days (number of days since 1st January 1981).                                                                                                                                                   |
| <b>ureal4 hist_start_time</b> | (in) Start time of history to retrieve in seconds since midnight.                                                                                                                                                                               |
| <b>uint2 num_hist</b>         | (in) Number of history values per point to be retrieved.                                                                                                                                                                                        |

|                               |                                                                                                                       |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>uint2 num_points</b>       | (in) Number of points to be processed. MAXIMUM value allowed is 20.                                                   |
| <b>uint2* point_type_nums</b> | (in) Array (of dimension num_points) containing the point type/numbers of the point history values to retrieve.       |
| <b>uint2* point_params</b>    | (in) Array of (dimension num_points) containing the parameter numbers of the history values to retrieve.              |
| <b>uchar* archive_path</b>    | This member is no longer in use and is only retained for backwards compatibility. Instead, pass a zero length string. |
| <b>real4* hist_values</b>     | (out) Array (of dimension num_points * num_hist) to provide storage for the returned history values.                  |
| <b>uint2 gethst_status</b>    | (out) Return value of the actual remote gethst_date call.                                                             |

The structure of the rgethst\_ofst\_data structure is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                               |                                                                                                                                                                                                                                                         |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>uint2 hist_type</b>        | (in) Defines the type of history to retrieve, this will be one of the standard server history types. Namely using one of the following defines:<br>HST_1MIN, HST_6MIN, HST_1HOUR, HST_8HOUR, HST_24HOUR, HST_5SECF, HST_1HOURE, HST_8HOURE, HST_24HOURE |
| <b>uint4 hist_offset</b>      | (in) Offset from latest history value in history intervals where offset=1 is the most recent history value).                                                                                                                                            |
| <b>uint2 num_hist</b>         | (in) Number of history values per point to be retrieved.                                                                                                                                                                                                |
| <b>uint2 num_points</b>       | (in) Number of points to be processed. MAXIMUM value allowed is 20.                                                                                                                                                                                     |
| <b>uint2* point_type_nums</b> | (in) Array (of dimension num_points) containing the point type/numbers of the point history values to retrieve.                                                                                                                                         |
| <b>uint2* point_params</b>    | (in) Array of (dimension num_points) containing the parameter numbers of the history values to retrieve.                                                                                                                                                |
| <b>uchar* archive_path</b>    | This member is no longer in use and is only retained for backwards compatibility. Instead, pass a zero length string.                                                                                                                                   |
| <b>real4* hist_values</b>     | (out) Array (of dimension num_points * num_hist) to provide storage for the returned history values.                                                                                                                                                    |
| <b>uint2 gethst_status</b>    | (out) Return value of the actual remote gethst_date call.                                                                                                                                                                                               |

The program using this function call must ensure that the size of the network packets generated does not exceed the maximum packet size permitted on the network. To meet this requirement, adhere to the following guideline:

- For request packets for `rhsc_param_hist_dates`:

```
(15 * number of history requests) + (2 * number of points requested)
+ string lengths of archive paths < 4000.
```

- For request packets for `rhsc_param_hist_offsets`:

```
(11 * number of history requests) + (2 * number of points requested)
+ string lengths of archive paths < 4000.
```

- For response packets:

```
(4 * number of history requests) + (4 * (For each history request
the sum of (num_hist * num_points)))
```

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

### rhsc\_param\_value\_puts

---

#### Attention:

**rhsc\_param\_value\_puts** is deprecated and may be removed in a future release. It is provided compatibility purposes only. When used with an Experion server release R400 or later **rhsc\_param\_value\_puts** will only be able to access points in the range:  $1 \leq \text{point number} \leq 65,000$ . The replacement function **rhsc\_param\_value\_puts\_2** should be used instead.

---

Control a list of point parameter values.

## C Synopsis

```
int rhsc_param_value_puts
(
    char*          szHostname,
    int            cprmvd,
    PARAM_VALUE_DATA* rgprmvd
);
```

## VB Synopsis

```
rhsc_param_value_puts (ByVal hostname As String,
    ByVal num_requests As Long,
    Param_value_data_array ()
    As param_value_data) As Long
```

## Arguments

| Argument          | Description                                                                                    |
|-------------------|------------------------------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server that the database resides on                                               |
| <b>cprmvd</b>     | (in) The number of controls to parameters requested                                            |
| <b>rgprmvd</b>    | (in/out) Pointer to a series of PARAM_VALUE_DATA structures (one array element for each point) |

## Description

The structure of the PARAM\_VALUE\_DATA structure is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                            |                              |
|----------------------------|------------------------------|
| <b>n_ushort nPnt</b>       | (in) point number            |
| <b>n_ushort nPrm</b>       | (in) parameter number        |
| <b>n_long nPrmOffset</b>   | (in) point parameter offset  |
| <b>PARvalue* pupvValue</b> | (in) parameter value union   |
| <b>n_ushort nType</b>      | (in) value type              |
| <b>n_long fStatus</b>      | (out) status of each request |

RHSC\_PARM\_VALUE\_PUTS writes a list of point parameter values to the specified remote server and performs the necessary control. A function return of 0 is given if the point parameter values are successfully controlled, otherwise, an error code is returned.

You can write a list of parameter values with different types using a single request. The value is placed into a union (of type PARvalue). Before storing the value to be written to a point parameter in the PARAM\_VALUE\_DATA structure, you must allocate sufficient memory for the union. You must free this memory before exiting your network application.

Although this is a list based function, there is no implication that it should be used as a sequential write function. If any individual put fails, the function will not prevent the remaining writes from occurring. The function will instead continue to write values to the remaining point parameters in the list.

Be careful when using `rhsc_param_value_puts()` and `rhsc_param_value_put_bynames()` with more than one point/parameter pair. Each put causes a control to be executed on the server and each control takes a small amount of time. If more than one pair is put, the total time for each of these controls may exceed the default TCP/IP timeout. This will cause the Network API to report the error `RCV_TIMEOUT`, even though all puts may have been successful. In addition, the Network API will be unavailable until the list of puts has been processed. This could cause subsequent calls to the network API to fail until the list is processed.

To simplify the handling of enumerations, two data types have been included for use with this function only. The data types are `DT_ENUM_ORD`, and `DT_ENUM_STR`. When writing a value to an enumeration point parameter, supply the ordinal part of the enumeration only and use the `DT_ENUM_ORD` data type. Alternatively, if you don't know the ordinal value, supply only the text component of the enumeration and use the `DT_ENUM_STR` data type. If the `DT_ENUM` data type is specified, only the ordinal value is used by this function (similar to `DT_ENUM_ORD`).

A successful return status from the **`rhsc_param_value_puts`** call indicates that no network errors were encountered (that is, the request was received, processed, and responded to).

If the returned value is `NADS_PARTIAL_FUNC_FAIL`, then at least one request (and possibly all requests) failed. The status field of each array element should be checked to find which request failed. For each array element, a value of `CTLOK` (See *Diagnostics for Network API functions* on page 375) or 0 in the status field indicates that the control was successful.

The program using this function call must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network.

Due to the ability to control a list of parameters of mixed data type, it is difficult to give generic limits. To meet the program requirement not to exceed the maximum packet size permitted, adhere to the following guidelines given for a number of specific cases:

- For request packets when writing `DT_INT2` data only:  
(12 \* number of points parameters) < 4000
- For request packets when writing `DT_INT4` data only:  
(14 \* number of points parameters) < 4000
- For request packets when writing `DT_REAL` data only:  
(14 \* number of points parameters) < 4000
- For request packets when writing `DT_DBLE` data only:  
(18 \* number of points parameters) < 4000
- For request packets when writing `DT_CHAR` data only:

(11 \* number of points parameters) + sum of string lengths of value character strings in bytes < 4000

- For request packets when writing DT\_ENUM\_ORD data only:

(11 \* number of points parameters) < 4000

- For request packets when writing DT\_ENUM\_STR data only:

(11 \* number of points parameters) + sum of string lengths of value enumeration strings in bytes < 4000

- For ALL reply packets:

(4 \* number of point parameters) < 4000

## Example

Control pntanal's SP value to 42.0 and change its DESC to say 'Funky description.'

```
int      status;
int      i;
POINT_NUMBER_DATA   rgpntnd[] = {{'pntanal'}};
PARAM_NUMBER_DATA   rgprmnd[] = {{0, 'SP'}, {0, 'DESC'}};

#define cpntnd sizeof(rgpntnd)/sizeof(POINT_NUMBER_DATA)
#define cprmnd sizeof(rgprmnd)/sizeof(PARAM_NUMBER_DATA)
/* There are the same number of PARAM_VALUE_DATA entries as cprmnd. */
#define cprmvd sizeof(rgprmnd)/sizeof(PARAM_NUMBER_DATA)

PARAM_VALUE_DATA   rgprmvd[cprmvd];

status = rhsc_point_numbers("Server1", cpntnd, rgpntnd);

rgprmnd[0].nPnt = rgpntnd[0].nPnt;
rgprmnd[1].nPnt = rgpntnd[0].nPnt;
status = rhsc_param_numbers("Server1", cprmnd, rgprmnd);

/* Set the point number, parameter number and offset for the point parameter. Allocate space, assign a value, and set the type for pntanal.PV */
rgprmvd[0].nPnt = rgprmnd[0].nPnt;
rgprmvd[0].nPrm = rgprmnd[0].nPrm;
rgprmvd[0].nPrmoffset = 1 /* Set parameter offset to default value*/
```

```

rgprmvd[0].pupvValue = (PARvalue *)malloc(sizeof(DT_REAL));
rgprmvd[0].pupvValue->real = (float)42.0;
rgprmvd[0].nType = DT_REAL;

/* Set the point number, parameter number and offset for the point parameter. Allocate space, assign a value, and set the type for pntanal.DESC */
rgprmvd[1].nPnt = rgprmd[1].nPnt;
rgprmvd[1].nPrm = rgprmd[1].nPrm;
rgprmvd[1].nPrmoffset = 1 /* Set parameter offset to default value*/
rgprmvd[1].pupvValue =
    (PARvalue *)malloc(strlen('Funky description') + 1); strcpy(rgprmvd
[1].pupvValue->text, 'Funky description');
rgprmvd[1].nType = DT_CHAR;

status = rhsc_param_value_puts('Server1', cprmvd, rgprmvd);
switch (status)
{
    case 0:
        printf('rhsc_param_value_puts successful\n');
        break;
    case NADS_PARTIAL_FUNC_FAIL:
        printf('rhsc_param_value_puts partially failed\n');
        /* Check fStatus flags to find out which ones failed. */
        break;
    default:
        printf('rhsc_param_value_puts failed(c_geterrno() = 0x%x)\n',status);
        break;
}

for (i=0; i<cprmvd; i++)
{
    free(rgprmvd[i].pupvValue);
}

```

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## See also

*rhsc\_param\_values* below

*rhsc\_param\_value\_put\_bynames* on page 318

## rhsc\_param\_values

### Attention:

**rhsc\_param\_values** is deprecated and may be removed in a future release. It is provided compatibility purposes only. When used with an Experion server release R400 or later **rhsc\_param\_values** will only be able to access points in the range:  $1 \leq \text{point number} \leq 65,000$ . The replacement function **rhsc\_param\_values\_2** should be used instead.

Read a list of point parameter values.

## C Synopsis

```
int rhsc_param_values
(
    char*          szHostname,
    int            nPeriod,
    int            cprmvd,
    PARAM_VALUE_DATA* rgprmvd
);
```

## VB Synopsis

```
rhsc_param_values (ByVal hostname As String,
    ByVal period as Long,
    ByVal num_requests as Long,
    param_value_data_array()
    As param_value_data) As Long
```

## Arguments

| Argument          | Description                                                                                                                                                 |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server that the database resides on.                                                                                                           |
| <b>nPeriod</b>    | (in) subscription period in milliseconds for the point parameters. Use the constant NADS_READ_CACHE if subscription is not required. If the value is in the |

| Argument       | Description                                                                                                                                                                                                                                                                     |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                | Experion cache, then that value will be returned. Otherwise the controller will be polled for the latest value. Use the constant NADS_READ_DEVICE if you want to force Experion to re-poll the controller. The subscription period will not be applied to standard point types. |
| <b>cprmvd</b>  | (in) The number of parameter values requested.                                                                                                                                                                                                                                  |
| <b>rgprmvd</b> | (in/out) Pointer to an array of PARAM_VALUE_DATA structures (one array element for each request).                                                                                                                                                                               |

## Description

The structure of the PARAM\_VALUE\_DATA structure is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                            |                              |
|----------------------------|------------------------------|
| <b>n_ushort nPnt</b>       | (in) point number            |
| <b>n_ushort nPrm</b>       | (in) parameter number        |
| <b>n_long nPrmOffset</b>   | (in) point parameter offset  |
| <b>PARvalue* pupvValue</b> | (out) parameter value union  |
| <b>n_ushort nType</b>      | (out) value type             |
| <b>n_long fStatus</b>      | (out) status of each request |

If your system uses *dynamic scanning*, `rhsc_param_values` calls from the Network API do not trigger dynamic scanning.

RHSC\_PARM\_VALUES requests a list of point parameter values from the specified remote server. A function return of 0 is given if the parameter values were successfully read else an error code is returned.

You can read a list of parameter values with different types using a single request. Each point parameter value is placed into a union (of type PARvalue). Before making the request, you must allocate sufficient memory for each value union. You must free this memory before exiting your Network application.

A successful return status from the **rhsc\_param\_values** call indicates that no network errors were encountered (that is, the request was received, processed, and responded to). If the returned value is NADS\_PARTIAL\_FUNC\_FAIL, then at least one request (and possibly all requests) failed. The status field of each array element should be checked to find which request failed.

The program using this function call must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network.

Due to the ability to acquire a list of parameters of mixed data type, it is difficult to give generic limits. To meet the program requirement not to exceed the maximum packet size permitted, adhere to the following guidelines given for a number of specific cases:

- For ALL request packets:  
(8 \* number of point parameters) < 4000
- For response packets when reading DT\_INT2 data only:  
(8 \* number of points parameters) < 4000
- For response packets when reading DT\_INT4 data only:  
(10 \* number of points parameters) < 4000
- For response packets when reading DT\_REAL data only:  
(10 \* number of points parameters) < 4000
- For response packets when reading DT\_DBLE data only:  
(14 \* number of points parameters) < 4000
- For response packets when reading DT\_CHAR data only:  
(7 \* number of points parameters) + sum of string lengths of value character strings in bytes < 4000
- For response packets when reading DT\_ENUM data only:  
(11 \* number of points parameters) + sum of string lengths of value enumeration strings in bytes < 4000

## Example

Read the value of pntana1.SP and pntana1.DESC.

```
int    status;
int    i;
POINT_NUMBER_DATA  rgpntnd[] = {{'pntana1'}};
PARAM_NUMBER_DATA  rgprmnd[] = {{0, 'SP'}, {0, 'DESC'}};

#define cpntnd sizeof(rgpntnd)/sizeof(POINT_NUMBER_DATA)
#define cprmnd sizeof(rgprmnd)/sizeof(PARAM_NUMBER_DATA)
/* There are the same number of PARAM_VALUE_DATA entries as cprmnd. */
```

---

```

#define cprmvd sizeof(rgprmnd)/sizeof(PARAM_NUMBER_DATA)

PARAM_VALUE_DATA  rgprmvd[cprmvd];

status = rhsc_point_numbers("server1", cpntnd, rgpntnd);
rgprmnd[0].nPnt = rgpntnd[0].nPnt;
rgprmnd[1].nPnt = rgpntnd[0].nPnt;
status = rhsc_param_numbers("server1", cprmnd, rgprmnd);

for (i=0; i<cprmvd; i++)
{
    rgprmvd[i].nPnt = rgprmnd[i].nPnt;
    rgprmvd[i].nPrm = rgprmnd[i].nPrm;
    /*Use of the parameter offset is currently unsupported. Set offset to
the default value 1. */
    rgprmvd[i].nPrmOffset = 1;
}

/*
ALLOCATING MEMORY:
Sufficient memory must be allocated for each value union. If the
value type is not known, allocate memory for the largest possible
size of a PARvalue union. See below for an example of how to allocate
this memory.
If the data type is known, then allocate the exact amount of memory
to save space.
For example for DT_REAL values:
    rgprmvd[0].pupvValue = (PARvalue *) malloc(sizeof(DT_REAL));
*/
for (i=0; i<cprmvd; i++)
{
    rgprmvd[i].pupvValue = (PARvalue *)malloc(sizeof(PARvalue));
}

status = rhsc_param_values('server1',
NADS_READ_CACHE, cprmvd, rgprmvd);

```

---

```
switch (status)
{
    case 0:
        printf('rhsc_param_values successful\n');
        for (i=0; i<cprmvd; i++)
        {
            switch (rgprmvd[i].nType)
            {
                case DT_CHAR:
                    printf('%s.%s has the value %s\n',
                        rgpntnd[0].szPntName,
                        rgprmnd[i].szPrmName,
                        rgprmvd[i].pupvValue->text);
                    break;
                case DT_INT2:
                    printf('%s.%s has the value %d\n',
                        rgpntnd[0].szPntName,
                        rgprmnd[i].szPrmName,
                        rgprmvd[i].pupvValue->int2);
                    break;
                case DT_INT4:
                    printf('%s.%s has the value %d\n',
                        rgpntnd[0].szPntName,
                        rgprmnd[i].szPrmName,
                        rgprmvd[i].pupvValue->int4);
                    break;
                case DT_REAL:
                    printf('%s.%s has the value %f\n',
                        rgpntnd[0].szPntName,
                        rgprmnd[i].szPrmName,
                        rgprmvd[i].pupvValue->real);
                    break;
                case DT_DBLE:
                    printf('%s.%s has the value %f\n',
                        rgpntnd[0].szPntName,
```

---

---

```

        rgprmnd[i].szPrmName,
        rgprmvd[i].pupvValue->dbld);
        break;
    case DT_ENUM:
        printf('%s.%s has the ordinal value
        %d and enum string %s\n',
        rgpntnd[0].szPntName,
        rgprmnd[i].szPrmName,
        rgprmvd[i].pupvValue->en.ord,
        rgprmvd[i].pupvValue->en.text);
        break;
    default:
        printf('Illegal type found\n');
        break;
    }
}
break;
case NADS_PARTIAL_FUNC_FAIL:
    printf('rhsc_param_values partially failed\n');
    /* Check fStatus flags to find out which ones failed. */
    break;
default:
    printf('rhsc_param_values failed (c_geterrno() = 0x%x)\n', status);
break;
}

for (i=0; i<cprmvd; i++)
{
    free(rgprmvd[i].pupvValue);
}

```

---

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## See also

*rhsc\_param\_value\_puts* on page 355

*rhsc\_param\_value\_put\_bynames* on page 318

## rhsc\_param\_numbers

### Attention:

**rhsc\_param\_numbers** is deprecated and may be removed in a future release. It is provided compatibility purposes only. When used with an Experion server release R400 or later **rhsc\_param\_numbers** will only be able to access points in the range: 1 <= point number <= 65,000. The replacement function **rhsc\_param\_numbers\_2** should be used instead.

Resolve a list of parameter names to numbers.

### C Synopsis

```
int rhsc_param_numbers(char* szHostname,
    int cprmnd,
    PARAM_NUMBER_DATA* rgprmnd);
```

### VB Synopsis

```
rhsc_param_numbers(ByVal hostname As String,
    ByVal num_requests As Long,
    param_number_data_array() As
    param_number_data) As Long
```

### Arguments

| Argument          | Description                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server on which the database resides                                           |
| <b>cprmnd</b>     | (in) The number of parameter name resolutions requested                                     |
| <b>rgprmnd</b>    | (in/out) Pointer to an array of PARAM_NUMBER_DATA structures (one for each point parameter) |

### Description

The structure of the PARAM\_NUMBER\_DATA structure is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                          |                                |
|--------------------------|--------------------------------|
| <b>n_ushort nPnt</b>     | (in) point number              |
| <b>n_char* szPrmName</b> | (in) parameter name to resolve |

|                       |                                 |
|-----------------------|---------------------------------|
| <b>n_ushort nPrm</b>  | (out) parameter number returned |
| <b>n_long fStatus</b> | (out) status of each request    |

RHSC\_PARAM\_NUMBERS converts a list of point parameter names to their equivalent parameter numbers for a specified remote server.

A successful return status from the `rhsc_param_numbers` call indicates that no network errors were encountered (that is, the request was received, processed, and responded to). If the returned value is `NADS_PARTIAL_FUNC_FAIL`, then at least one request (and possibly all requests) failed. The status field of each array element should be checked to find which request failed.

The program using this function call must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network. To meet this requirement, adhere to the following guidelines:

- For request packets:

$(4 * \text{number of points}) + \text{sum of string lengths of point names in bytes} < 4000$

- For response packets:

$(6 * \text{number of points}) < 4000$

---

## Example

Resolve the parameter names 'pntana1.SP' and 'pntana1.DESC.'

```
int    status;
int    i;
POINT_NUMBER_DATA  rgpntnd[] = {{'pntana1'}};
PARAM_NUMBER_DATA  rgprmnd[] = {{0, 'SP'}, {0, 'DESC'}};

#define cpntnd sizeof(rgpntnd)/sizeof(POINT_NUMBER_DATA)
#define cprmnd sizeof(rgprmnd)/sizeof(PARAM_NUMBER_DATA)

status = rhsc_point_numbers('server1', cpntnd, rgpntnd);
/* Check for error status. */

/* Grab the point numbers from the rgpntnd array. */
rgprmnd[0].nPnt = rgpntnd[0].nPnt;
rgprmnd[1].nPnt = rgpntnd[0].nPnt;
```

---

```
status = rhsc_param_numbers('server1', cprmnd, rgprmnd);
switch (status)
{
    case 0:
        printf('rhsc_param_numbers successful\n');
        for (i=0; i<cprmnd; i++)
        {
            printf('%s.%s has the parameter number %d\n',
                rgpntnd[0].szPntName,
                rgprmnd[i].szPrmName,
                rgprmnd[i].nPrm);
        }
    case NADS_PARTIAL_FUNC_FAIL:
        printf('rhsc_param_numbers partially failed\n');
        /* Check fStatus flags to find out which ones failed. */
        break;
    default:
        printf('rhsc_param_numbers failed (c_geterrno() = 0x%x)\n', status);
        break;
}
```

---

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

## See also

*rhsc\_point\_numbers* below

## rhsc\_point\_numbers

---

### Attention:

**rhsc\_point\_numbers** is deprecated and may be removed in a future release. It is provided compatibility purposes only. When used with an Experion server release

---

R400 or later **rhsc\_point\_numbers** will only be able to access points in the range: 1<= point number <= 65,000. The replacement function **rhsc\_point\_numbers\_2** should be used instead.

Resolve a list of point names to numbers.

### C/C++ Synopsis

```
int rhsc_point_numbers
(
    char*          szHostname,
    int            cpntnd,
    POINT_NUMBER_DATA*  rgpntnd
);
```

### VB Synopsis

```
rhsc_point_numbers(ByVal hostname As String,
    ByVal num_requests As Long,
    POINT_NUMBER_DATA_array()
    As POINT_NUMBER_DATA) As Long
```

### Arguments

| Argument          | Description                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------|
| <b>szHostname</b> | (in) Name of server that the database resides on                                                  |
| <b>cpntnd</b>     | (in) The number of point name resolutions requested                                               |
| <b>rgpntnd</b>    | (in/out) Pointer to a series of POINT_NUMBER_DATA structures (one array element for each request) |

### Description

The structure of the POINT\_NUMBER\_DATA structure is defined in nif\_types.h. This structure and its members are defined as follows:

|                          |                              |
|--------------------------|------------------------------|
| <b>n_char* szPntName</b> | (in) point name to resolve   |
| <b>n_ushort nPnt</b>     | (out) point number           |
| <b>n_long fStatus</b>    | (out) status of each request |

**RHSC\_POINT\_NUMBERS** converts a list of point names to their equivalent point numbers for a specified remote server.

A successful return status from the **rhsc\_point\_numbers** call indicates that no network errors were encountered (that is, the request was received, processed, and responded to). If the returned value is **NADS\_PARTIAL\_FUNC\_FAIL**, then at least one request (and possibly all requests) failed. The status field of each array element should be checked to find which request failed.

The program using this function call must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network. To meet this program requirement, adhere to the following guidelines:

- For request packets:

(2 \* number of points) + sum of string lengths of point names in bytes < 4000

- For response packets:

(6 \* number of points) < 4000

### Example

Resolve the point names 'pntana1' and 'pntana2'.

```
int     status;
int     i;
POINT_NUMBER_DATA  rgpntnd[] = {'pntana1'},{'pntana2'};
#define cpntnd sizeof(rgpntnd)/sizeof(POINT_NUMBER_DATA)

status = rhsc_point_numbers('Server1', cpntnd, rgpntnd);

switch (status)
{
    case 0:
        printf('rhsc_point_numbers successful\n');
        for (i=0; i<cpntnd; i++)
        {
            printf('%s has the point number %d\n',
                rgpntnd[i].szPntName,
                rgpntnd[i].nPnt);
        }
        break;
}
```

```

    case NADS_PARTIAL_FUNC_FAIL:
        printf('rhsc_point_numbers partially failed\n');
        /* Check fStatus flags to find out which ones failed. */
        break;
    default:
        printf('rhsc_point_numbers failed (c_geterrno() = 0x%x)\n', status);
        break;
}

```

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

### rputpntval

Set the numeric parameter value.

This function's synopsis and description are identical to that of 'rputpntval\_ascii.'

### rputpntval\_ascii

Set the ASCII parameter value.

## VB Synopsis

```

rputpntval (ByVal server As String,
            ByVal point As String,
            ByVal param As Integer,
            value As Single) As Integer
rputpntval_ascii (ByVal server As String,
                  ByVal point As String,
                  ByVal param As Integer,
                  value As String) As Integer

```

## Arguments

| Argument      | Description                                      |
|---------------|--------------------------------------------------|
| <b>server</b> | (in) Name of server that the database resides on |
| <b>point</b>  | (in) Name of point                               |
| <b>param</b>  | (in) Point parameter number                      |

| Argument | Description                                         |
|----------|-----------------------------------------------------|
| value    | (out) Value of point parameter returned by function |

## Description

This function is provided for backwards compatibility. It cannot be used to access point information for points on Process Controllers. This function can only access points in the range: 1 <= point number <=65,000.

## Diagnostics

See *Diagnostics for Network API functions* on page 375.

### rputval\_hist

Store history values.

This function's synopsis and description are identical to that of 'rputval\_ascii.'

### rputval\_num

Store the value of numeric point parameters.

This function's synopsis and description are identical to that of 'rputval\_ascii.'

### rputval\_ascii

Store the value of ASCII point parameters.

## C/C++ Synopsis

```
int rputval_num
(
    char*   server,
    int     num_points,
    rputval_num_data*  putval_num_data
);
int rputval_ascii
(
    char*   server,
    int     num_points,
    rputval_ascii_data*  putval_ascii_data
);
int rputval_hist
```

```
(
    char*    server,
    int      num_points,
    rputval_hist_data*  putval_hist_data
);
```

### VB Synopsis

```
rputval_numb(ByVal server As String,
    ByVal num_points As Integer,
    putval_numb_data() As rputval_numb_data_str)
    As Integer
rputval_ascii(ByVal server As String,
    ByVal num_points As Integer,
    putval_ascii_data() As rputval_ascii_data_str)
    As Integer
rputval_hist (ByVal server As String,
    ByVal num_points As Integer
    putval_hist_data() As rputval_hist_str)
    As Integer
```

### Arguments

| Argument                | Description                                                                       |
|-------------------------|-----------------------------------------------------------------------------------|
| <b>server</b>           | (in) Name of server that the database resides on                                  |
| <b>num_points</b>       | (in) The number of points passed to rputval_xxxx in the putval_xxxx_data argument |
| <b>putval_xxxx_data</b> | (in/out) Pointer to a series of rputval_xxxx_data structures (one for each point) |

### Description

This function is provided for backwards compatibility. It cannot be used to access point information for points on Process Controllers. This function can only access points in the range: 1 <= point number <=65,000.

The rhsc\_param\_value\_puts\_2 function should be used instead.

This function call enables access to point parameter values. The three types of parameters (numerical, ASCII, and history) are accessed using the corresponding function variations. The point parameters to be accessed are referenced by the members of the rputval\_numb\_data, rputval\_ascii\_data, and rputval\_hist\_data structures (see below). The functions accept an array of structures, thus providing access to multiple point parameter values with one call.

The structure of the `rputval_num_data` structure is defined in **nif\_types.h**. This structure and its members are defined as follows:

|                                         |                                                                                                                                                                                                        |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>n_ushort<br/>point_<br/>type_num</b> | (in) Defines the point type/number to be accessed.                                                                                                                                                     |
| <b>n_ushort<br/>point_<br/>param</b>    | (in) Defines the point parameter to be accessed (for example, process variable (PV), Mode (MD), output (OP) or set point (SP)). The definitions for parameter type are located in the parameters file. |
| <b>n_float<br/>param_<br/>value</b>     | (out) Value of point parameter to be stored.                                                                                                                                                           |
| <b>n_short<br/>putval_<br/>status</b>   | (out) The return value of the actual remote putval call.                                                                                                                                               |

The structure of the `rputval_ascii_data` structure is defined in **nif\_types.h**. This structure and its members is defined as follows:

|                                         |                                                                                                                                                           |
|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>n_ushort<br/>point_type_<br/>num</b> | (in) Defines the point type/number to be accessed.                                                                                                        |
| <b>n_ushort<br/>point_param</b>         | (in) Defines the point parameter to be accessed (for example, description (DESC)). The definitions for parameter type are located in the parameters file. |
| <b>n_char*<br/>param_value</b>          | (in) ASCII string value of point parameter to be stored (Note this does not need to be null terminated).                                                  |
| <b>uint2<br/>param_len</b>              | (in) Length of above param_value to be stored (in bytes).                                                                                                 |
| <b>n_ushort<br/>putval_status</b>       | (out) The return value of the actual remote putval call.                                                                                                  |

The structure of the `rputval_hist_data` structure is defined in **nif\_types.h**. This structure and its members is defined as follows:

|                                      |                                                                                                                                                                    |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>uint2<br/>point_<br/>type_num</b> | (in) Defines the point type/number to be accessed.                                                                                                                 |
| <b>uint2<br/>point_<br/>param</b>    | (in) Defines the point parameter to be accessed (for example, 1 minute history (HST_1MIN)). The definitions for parameter type are located in the parameters file. |
| <b>uint2 hist_</b>                   | (in) Offset from latest history value in history intervals to store value (Where hist_                                                                             |

|                            |                                                          |
|----------------------------|----------------------------------------------------------|
| <b>offset</b>              | offset=1 is the most recent history value).              |
| <b>real4 param_value</b>   | (in) Value of point parameter to be stored.              |
| <b>uint2 putval_status</b> | (out) The return value of the actual remote putval call. |

Note that a successful return status from the **rputval** call indicates that no network error was encountered (that is, the request was received, processed, and responded to). The status field in each call structure needs to be verified on return to determine the result of the individual remote calls.

The program using these function calls must ensure that the size of the network packets generated do not exceed the maximum packet size permitted on the network. This requirement can be met by adhering to the following guideline:

`(12 * number of points) + sum of all string value lengths in bytes <4000`

## Diagnostics

See *Diagnostics for Network API functions* below.

### Diagnostics for Network API functions

Unless otherwise stated, all Network API functions behave as follows: upon successful completion, a value of 0 is returned; otherwise one or more of the following error codes is returned.

#### CTLOK (0x8220)

This is not actually an error code, but an indication that the control was executed successfully.

#### GHT\_HOST\_TABLE\_FULL (0x8808)

The API cannot store further information about host systems.

#### M4\_CDA\_ERROR (0x8155)

The CDA subsystem has reported an error. The two most likely causes are that the CDA service has been stopped on the primary server, or you have attempted to write to a read-only process point.

#### M4\_CDA\_WARNING (0x8156)

The CDA subsystem has reported a warning.

#### M4\_DEVICE\_TIMEOUT (0x106)

There has been a timeout when communicating to a field device. This may occur when

attempting to read from a process point parameter if the CDA service has been stopped on the primary server. It may also occur if a field device fails to respond at all, or before the timeout period, when performing a control.

**M4\_GDA\_COMMS\_ERROR (0x8153)**

There has been a communications error. See the log file for further details. This may occur when accessing a remote point if the remote server is offline or failing over. You may also see this error when accessing a flexible point.

**M4\_GDA\_COMMS\_WARNING (0x8154)**

There has been a communications warning.

**M4\_GDA\_ERROR (0x8150)**

There has been an error reported by the data access subsystem. See the log file for further details. This may occur when accessing a remote point (on another server) or a flexible point.

**M4\_INV\_PARAMETER (0x8232)**

There was an attempt to access a parameter either by name or by parameter number, but the point does not have a parameter by that name or number.

**M4\_INV\_POINT (0x8231)**

There was an attempt to access a point either by name or by point number, but that point name or number does not exist.

**M4\_PNT\_ON\_SCAN (0x8212)**

There was an attempt to write to a read-only parameter of a non-process point while it was on scan.

**M4\_SYSTEM\_OFFLINE (0x83fc)**

The system is offline.

**NADS\_ARRAY\_DIM\_ERROR (0x83A0)**

A VB array has been dimensioned with an incorrect number of dimensions. The API expects all arrays to be single dimensioned.

**NADS\_ARRAY\_INVALID\_ELEMENT\_SIZE (0x83A1)**

There is a mismatch between the size of the elements passed to the API and the size of elements expected by the API. Ensure that you have not modified any byte-alignment settings in Visual Basic.

**NADS\_ARRAY\_OVERFLOW (0x839F)**

A VB array passed to the API is not large enough to contain the information requested.

NADS\_AUTHORIZATION\_FAIL (0x83A6)

The user who is logged in has insufficient scope of responsibility to perform the request.

NADS\_BAD\_POINT\_PAR (0x838C)

A bad point parameter value was sent or received.

NADS\_CLOSE\_ERR (0x8394)

A network error occurred. The network socket could not be closed correctly.

NADS\_GLOBAL\_ALLOC\_FAIL (0x8396)

The system was unable to allocate enough memory to perform the requested operation. Close any unnecessary running application to free more memory.

NADS\_GLOBAL\_LOCK\_FAIL (0x8395)

An internal error occurred. The system was unable to access internal memory.

NADS\_HOST\_ER (0x8392)

The server name specified was not recognized. Check the **hosts** file and DNS settings.

NADS\_HOST\_MISMATCH (0x8388)

Retries exhausted and last reply was from the wrong host.

NADS\_HOST\_NOT\_PRIMARY (0x8398)

The host is in redundant backup mode.

NADS\_INCOMPLETE\_HEADER (0x8387)

Retries exhausted and last reply was a runt packet.

NADS\_INIT\_ER (0x8390)

An internal error occurred. The system was unable to initialize correctly. Restart your application.

NADS\_INVALID\_LIST\_SIZE (0x839B)

The number of requests specified when calling the function was less than 1 and is invalid.

NADS\_INVALID\_PROT (0x838E)

An internal error occurred. An unknown network protocol was specified.

NADS\_INVALID\_STATUS (0x8397)

An internal error occurred. The server returned an invalid status.

NADS\_NO\_DLL (0x838D)

An internal error occurred. No network dll could be found.

NADS\_NO\_SUCH\_FUNC (0x8384)

The remote server being contacted does not support the requested function.

NADS\_NO\_SUCH\_VERS (0x8383)

The remote server being contacted does not support the requested version for the function concerned.

NADS\_PARTIAL\_FUNC\_FAIL (0x839A)

Warning that at least one request (and possibly all requests) in the list has returned its status in error.

NADS\_PORT\_MISMATCH (0x8389)

Retries exhausted and last reply was from the wrong protocol port.

NADS\_RCV\_TIMEOUT (0x8386)

The request timed out while waiting for the reply. Check network connections and that the server is running.

NADS\_REQ\_COUNT\_MISMATCH (0x839C)

An internal error occurred. The number of requests sent by the Client and received by the Server do not match.

NADS\_RX\_BUFFER\_EMPTY (0x8382)

An internal error occurred. A pull primitive has failed due to the NADS Stream receive buffer being empty.

NADS\_RX\_ERROR (0x8393)

An internal error occurred. A message was not received.

NADS SOCK\_ER (0x8391)

An internal error occurred. The socket count could not be opened.

NADS\_TRANS\_ID\_MISMATCH (0x838A)

Retries exhausted and last reply was from an obsolete request.

NADS\_TX\_BUFFER\_FULL (0x8381)

A push primitive has failed due to the NADS Stream transmit buffer being full.

NADS\_TX\_ER (0x838F)

The API failed to transmit a message.

NADS\_VAR\_TYPE\_MISMATCH (0x839D)

The VARIANT data type used in VB does not match the requested type of the PARvalue union in C.

NADS\_WRITES\_DISABLED (0x83a4)

Writes via the Network API have been disabled. For more information, see **Disable writes via Network API** in “Security tab, server wide settings” in the *Server and Client Configuration Guide*.

NADS\_WRONG\_PROGRAM (0x8385)

The remote server being contacted has a NADS program number assignment other than that specified in the request.

NADS\_WRONG\_PORT (0x83A5)

A write request was received on the read port, or a read request was received on the write port.

### Errors Received During a Failover

You may receive the following errors during a manual or automatic failover:

- M4\_SYSTEM\_OFFLINE
- NADS\_HOST\_NOT\_PRIMARY

If you are accessing a remote point and the DSA system is undergoing a manual or automatic failover you may see M4\_GDA\_COMMS\_ERROR.

---

## Using Experion's Automation Objects

This chapter provides an overview of issues applicable to developing applications that use Experion's Automation Object Models.

Experion includes the following object models, each of which represents a particular aspect of the system.

### Server Automation Object Model

The Server Automation Object Model represents the server, points, events, and reports.

#### Notes

- In Visual Basic, choose **ProjectReferences**, and select **HSC Server Automation Model 1.0** from the list.
- The Server object is the only object that can be directly created with the Visual Basic CreateObject function or the 'New' keyword.
- You must only create one Server object and cache its existence, although you can make copies of it.
- The Server object can only be created on a server if the database is loaded.
- An application that uses the Server Automation model can be run as:
  - An application with an allocated LRN. This is subject to the same security measures as any other application.
  - A utility on the server. This requires physical access to the server.

#### Applicable documentation

See the *Server Scripting Reference*.

#### Example

This example shows how to create the Server object.

```
Set objServer = CreateObject("HSCAutomationServer.Server")
```

### HMIWeb Object Model

The HMIWeb Object Model represents Station and HMIWeb displays.

## Notes

- The Application object is the only object that can be directly created with the Visual Basic CreateObject function or the 'New' keyword.
- DSP displays are represented by the *Station Object Model* on page 384.

## Applicable documentation

See the *HMIWeb Display Building Guide*.

## Example

This example shows how to create the top-level object (Application) of the HMIWeb Object Model.

```
Set objStationApp = CreateObject("Station.Application")
```

Once created, the application can then control Station through the object variable **objStationApp**. For example, to instruct Station to call up a display called 'CompressorStatus', the application would use the following code.

```
objStationApp.CurrentPage = CompressorStatus
```

## Station Scripting Objects

Station Scripting Objects (SSOs) are ActiveX controls that attach Station-level scripts to a Station. SSOs are based on the *HMIWeb Object Model* on the previous page.

## Notes

- The sample Visual Basic project for an SSO **.vbp** provides the framework for implementing an SSO. (The project and associated components are zipped into **SSO\_Sample.zip**. This file is located in **Station\Samples**.)
- Every SSO must implement a detach method. See 'Implementing a Detach method.'

## Applicable documentation

See the *HMIWeb Display Building Guide*.

## Creating an SSO

### To create an SSO

1. Open the sample SSO project in Visual Basic.
2. Change the **Project name** to something appropriate, which is unique.

When you change the name, the **ProgID** also changes (The ProgID is of the form **ProjectName.ClassName**.) For example, if you change the project name to **OperStations**, the **ProgId** will change to **OperStations.clsSSO**.

3. Write your scripts.

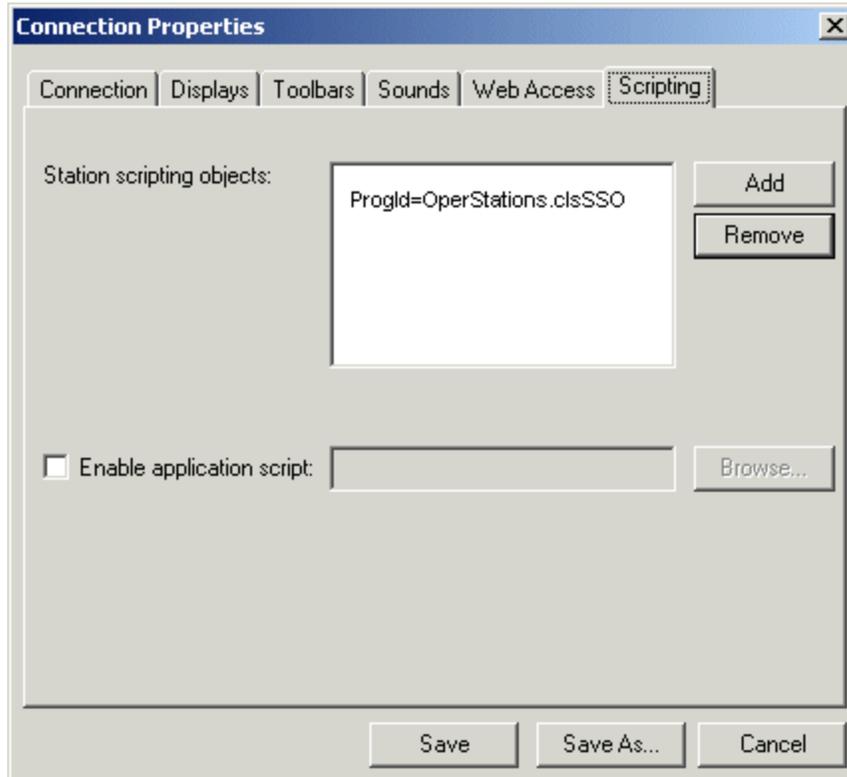
The sample SSO contains some simple code that adds entries to the Dictionary object, and displays a message box when Station connects to the server.

4. Compile the SSO by choosing **FileMake SSO.dll**. If there are any errors, you must fix them before moving onto the next step.
5. For each Station that needs the SSO:
  - a. Copy the SSO to the Station computer and register it. See *Registering an SSO* on the next page. (You can skip this step if you compiled the SSO on the computer because it automatically registers itself if the compilation is successful.)
  - b. Choose **StationConnection Properties**.

The **Connection Properties** dialog box opens.

- c. Click the **Scripting** tab.
- d. Click **Add** to add a new entry in the **Station scripting objects** list and type the

SSO's ProgID after '**ProgID** ='.



6. Click **Save** to save the changes to the connection.

### Registering an SSO

You must register an SSO on each Station computer that needs to use it. (Unless you have compiled it on the computer.)

### To register an SSO

1. Copy the SSO to the computer.
2. Open a Command Prompt window.
3. Type **regsvr32 SsoName .dll** , where **SsoName** is the name of the SSO (including its path).

### Implementing SetStation Object

The SetStationObject method must be implemented so that Station will create your SSO. As soon as Station has created your SSO, it calls SetStationObject, passing in a reference to Station's Application object.

You may use this method to initialize a global variable in your SSO that it references to Station's Application object.

---

### Visual Basic example:

```
Dim WithEvents m_objStation As Station.Application4
Public Sub SetStationObject(ByRef objStation As Station.Application4)
    Set m_objStation = objStation
End Sub
```

---

### Implementing a Detach method

An SSO must implement a detach method so that the SSO detaches correctly when Station exits. Include the following code in every SSO.

---

```
Public Sub Detach()
    Set m_objStation = Nothing
End Sub
```

---

## Station Object Model

The Station Object Model represents DSP displays.

### Notes

- Except for DSP displays and earlier versions of Station, the Station Object Model has been superseded by the *HMIWeb Object Model* on page 380.
- In order to control DSP displays, you must first create Station using the Application object of the *HMIWeb Object Model* on page 380 (see *HMIWeb Object Model* on page 380). After creating the Application object, you can then control DSP displays using the Station Object Model.

### Applicable documentation

See the *Display Building Guide*.

## Example

This example shows how to call up the Alarm Summary, a DSP display whose display number is **5**. (The variable, **objStationApp**, represents Station, which has already been created using the Application object of the HMIWeb Object Model.)

```
objStationApp.CurrentPage = 5
```

---

## Glossary

### accumulator point

A *point* type used to represent counters. Information contained in the accumulator point can include: the raw value, a process value, a rollover value, a scale factor, and a meter factor.

### acronym

An acronym is a text string that is used to represent a state or value of a *point* in a *display*. From an operator's point of view, it is much easier to understand the significance of an acronym such as 'Stopped', compared with an abstract value such as '0'.

### action algorithm

One of two types of algorithm you can assign to a point in order to perform additional processing to change point parameter values. An action algorithm performs an action when the value of the PV changes. Contrast with *PV algorithm*.

### ActiveX

COM-based technology for creating objects and components.

### ActiveX component

An ActiveX component is a type of program designed to be called up from other applications, rather than being executed independently. An example of an ActiveX component is a custom dialog box, which works in conjunction with *scripts*, to facilitate operator input into *Station*.

### Activity

A series of actions (with a start time and an end time) that occur in a plant. The term provides a market-neutral entity that can represent products such as batches, movement automation, and procedural operations.

### Activity Entity

An object from which an activity can be created (e.g., RCM or SCM for batch/procedure activities). Also known as *Recipe*.

### ADO

Active Data Object.

### alarm

An indication (visual and/or audible) that alerts an operator at a Station of an abnormal or critical condition. Each alarm has a type and a *priority*. Alarms can be assigned either to individual points or for system-wide conditions, such as a controller communications failure. Alarms can be viewed on a Station display and included in reports. Experion classifies alarms into the following types:

- PV Limit
- Unreasonable High and Unreasonable Low
- Control Failure
- External Change

alarm/event journal

A file that records all alarms and events. It is accessed to generate reports and can also be archived to off-line media.

alarm priority

One of **four** levels of severity specified for the alarm. The alarm priorities from least to most severe are:

- Journal
- Low
- High
- Urgent

Note that if critical alarm support has been enabled, urgent priority alarms with a sub priority of 15 will be shown as critical priority alarms on the Alarm Summary and other displays but the priority and subpriority will remain as configured for Experion APIs.

algorithm

See *point algorithm*.

analog point

A point type that is used to represent continuous values that are either real or integer. Continuous values in a process could be: pressure, flow, fill levels, or temperature.

ANSI

American National Standards Institute

API

Application Programming Interface

application program

A user-written program integrated into Experion using the Application Programming Interface (API).

asset

A logical sub-section of your plant or process. Custom displays, points, and access configuration may be associated with an asset. Operators and Stations can be assigned access to particular assets only.

automatic checkpointing

In a redundant server system, automatic checkpoint is the automatic transfer of database updates from the primary server to the backup server.

auxiliary parameter

An analog point parameter in addition to PV, SP, OP, and MD. Up to four auxiliary parameters can be used to read and write four related values without having to build extra points.

bad value

A parameter value, (for example, PV), that is indeterminate, and is the result of conditions such as unavailable input.

Client software

An umbrella term covering Experion Quick Builder, Station, and Display Builder software.

channel

The communications port used by the server to connect to a controller. Channels are defined using the Quick Builder tool.

CIM

Communications Interface Module

collection

A collection is a set of named values or *display objects* that are used in *scripts*.

COM

Component Object Model

control failure alarm

For *analog* and *status* points, an *alarm* configured to trigger when an OP, SP, MD, or a parameter control is issued and a demand scan on the source address, performed by the server, finds their value does not match the controlled value.

control level

A security designation assigned to a *point* that has a destination address configured (for analog or status points only). A control level can be any number from 0 to 255. An operator

will be able to control the point only if they have been assigned a control level equal to, or higher than, the point control level.

control parameters

Point parameters defined to be used as a control. A control parameter has both a source and a destination address. The destination for the parameter value is usually an address within the controller. Control parameters can be defined as automatic (server can change) or manual (operator can change).

controller

A device that is used to control and monitor one or more processes in field equipment. Controllers include Programmable Logic Controllers (PLCs), loop controllers, bar code readers, and scientific analyzers.

Controllers can be defined using the Quick Builder tool. Some controllers can be configured using Station displays.

database controller

See *User Scan Task controller*.

database point

Any *point* that has one or more parameters with database addresses.

DCD

Data Carry Detect

DCS

Digital Control System

DDE

Dynamic Data Exchange

default

The value that an application automatically selects if the user does not explicitly select another value.

deleted items

In Quick Builder, an item that has been flagged for deletion from the server database and appears in the Deleted grouping. When a download is performed, the item is deleted from both the server database and the Quick Builder project database.

demand scan

A one-time-only scan of a point parameter that can be requested either by an operator, a

report, or an application.

DHCP

Dynamic Host Configuration Protocol

display

Station uses displays to present Experion information to operators in a manner that they can understand. The style and complexity of displays varies according to the type of information being presented.

Displays are created in *Display Builder*.

Display Builder

The Honeywell tool for building customized graphical displays representing process data.

display object

A display object is a graphic element, such as an alphanumeric, a pushbutton or a rectangle, in a *display*.

Display objects that represent point information (such as an alphanumeric) or issue commands (such as a pushbutton) are called 'dynamic' display objects.

Distributed System Architecture (DSA)

An option that enables multiple Experion servers to share data, alarms, and history without the need for duplicate configuration on any server.

DNS

Domain Name System

DSR

Data Signal Ready

DTE

Data Terminal Equipment

DTR

Data Terminal Ready

dual-bit status point

A status point that reads two bits. Status points can read one, two or three bits.

EIM

Ethernet Interface Module

## ELPM

Ethernet Loop Processor Module

## EMI

Electromagnetic Interference

## event

A significant change in the status of an element of the system such as a point or piece of hardware. Some events have a low, high, or urgent priority, in which case they are further classified as alarms. Events can be viewed in *Station* and included in reports.

Within the context of *scripts* (created in *Display Builder* and used in *Station*), an event is a change in system status or an operator-initiated action that causes a *script* to run.

## Event Archiving

Event Archiving allows you to archive events to disk or tape, where they may be retrieved if needed.

## EXE

Executable.

## exception scan

A scan that takes place only when a change occurs at a controller address configured for a point parameter. Some controllers can notify the server when a change occurs within the controller. The server uses exception polling to interrogate the controller for these changes. This type of scan can be used to reduce the scanning load when a fast periodic scan is not required.

## export

In relation to *Station* displays, this refers to the process of registering a *display* with the *server* so that it can be called up in *Station*.

In relation to *Quick Builder*, this refers to the process of converting the configuration data in a project file into text files for use with other applications.

## Extended history

A type of history collection that provides snapshots of a point at a designated time interval that can be:

- 1-hour snapshots
- 8-hour snapshots
- 24-hour snapshots

### Fast history

An type of history that provides a 5-second snapshot history for points.

### field address

The address within the controller that contains stored information from a field device being monitored by the controller.

### free format report

An optional report type that enables users to generate their own report.

### FTP

File Transfer Protocol

### group

A group of up to eight related points whose main parameter values appear in the same group display. Sometimes called 'operating group'.

### history

Point values stored to enable tracking and observation of long-term trends. Analog, status, and accumulator point PVs can be defined to have history collected for them. Three types of history collection are available:

- Standard
- Extended
- Fast

### history gate

A status point parameter that is used to control the collection of history for an analog or status point. The history is only collected if the gate state value of the nominated parameter is in the nominated state.

### host server

In a DSA system, the server on which a remote point's definition is stored and from which alarms form the point originate.

### HTTP

Hypertext Transfer Protocol

### IDE

Integrated Development Environment.

## IEEE

Institute of Electrical and Electronic Engineers

## input value

Values that are usually scanned from the *controller* registers but can be from other server addresses. Input values can represent eight discrete states. Up to three values can be read from an address in order to determine a state.

## IRQ

Interrupt Request

## item

In Quick Builder, the elements necessary for data acquisition and control that comprise the Experion server data and are defined in the project file. These are:

- Channels
- Controllers
- Stations
- Points
- Printers

## item grouping

A collection of items grouped by a common property.

## item list

In Quick Builder, a listing of the items defined in the project file that displays in every Project View. The item list can be used to find an item and then display its properties.

## item number

Item numbers are used in the server database to identify items. In Quick Builder, the number is assigned to an item internally. The item numbers for channels, controllers, Stations and printers can be overwritten in Quick Builder to match an existing system database.

## local display object

A dynamic *display object* that displays information or issues a command, but which is not linked to the *server*. Such display objects are used in conjunction with *scripts*.

## local server

The server to which the Station is connected.

## MCI

Media Control Interface

## MD

Experion abbreviation for *mode*.

## method

A programmatic means of controlling or interrogating the *Station Automation object model*. A method is equivalent to the terms 'function' or 'command' used in some programming languages.

## Microsoft Excel Data Exchange

A network option that can be used to capture the most recent point and history information in the server and display it in Microsoft Excel spreadsheets, primarily for reporting.

## Mode

A point parameter which determines whether or not the operator can control the point value. For example, in a status point, the mode determines whether the operator can control the output value, and in an analog point the mode determines the control of the set point. If the mode is set to manual, the operator can change the value.

## Network Node controller

A server running the system software defined as a controller to another server running the system software. The local server can scan and control points that have been defined in the remote Network Node controller as long as those points have also been defined in the local server. The Network Node option is provided for backward compatibility.

## ODBC

See *Open Database Connectivity*.

## ODBC driver

A driver that processes ODBC (Open Database Connectivity) calls, queries the database, and returns the results. See also *Open Database Connectivity*.

## OP

Experion abbreviation for *output*.

## OPC

OPC stands for OLE (Object Linking & Embedding) for Process Control. It is a set of standards to facilitate interoperability between applications within the Process Control Industry. These include automation/control applications, field systems/devices or business/office applications.

OPC specifies a standard interface to be used between two types of applications called OPC clients and OPC servers. An OPC server is an application which collects data, generally directly from a physical device, and makes it accessible through the OPC interface. An OPC client requests and uses the data provided by an OPC Server. By having a standard interface OPC clients and servers written by different vendors can communicate.

#### Open Database Connectivity

A standard set of function calls for accessing data in a database. These calls include the facility to make SQL (Structured Query Language) queries on the database. To use ODBC you must have support from the client application (for example, Microsoft Access) which will generate the ODBC calls and from some database-specific software called an *ODBC driver*.

#### operator ID

A unique identification assigned to each operator. If Operator-Based security is enabled, the operator must use this ID and a password to sign on to a Station.

#### operator password

A character string (not echoed on screen) used with the operator ID to sign on to a Station.

#### operator security level

See *security level*.

#### Operator-based security

Operator-based security comprises an operator ID and password, which must be entered at a Station in order to access Experion functions.

#### output

A *point* parameter used to issue control values. The output (OP) is often related to the mode (MD) parameter and can be changed by an operator only if the mode is manual.

#### parameter

The different types of values accessed by *points* are known in Experion as 'point parameters.'

Experion can store and manage multiple values in the one point. You can therefore use a single point to monitor and control a complete loop.

The names of the parameters reflect their most common usage. They can, however, be used to hold any controller values.

#### periodic scan

A defined regular interval in which the server acquires information from the controller and processes the value as a point parameter. The scan period must be defined in Quick Builder for each point source parameter value.

PIN

Plant Information Network

PLC

Programmable logic controller

point

A data structure in the server database, usually containing information about a field entity. A point can contain one or more parameters.

point algorithm

A prescribed set of well-defined rules used to enhance a point's functionality. The point algorithm accomplishes this by operating on the point data either before or after normal point processing.

There are two types of point algorithms, PV (processed every time the point parameter is scanned) and Action (processed only when a point parameter value changes).

point detail display

A display that shows the current point information. Each point has a Point Detail display.

primary server

This is the PC that normally runs the database software, performs processing tasks, and allocates resources to client PCs. If the primary server is unavailable, the secondary server takes over until it is available again.

process variable

An actual value in a process: a temperature, flow, pressure, and so on. Process variables may be sourced from another parameter and may also be calculated from two or more measured or calculated variables using algorithms.

programmable logic controller (PLC)

A control and monitoring unit that connects to a field device and controls low-level plant processes with very high-speed responses. A PLC usually has an internal program that scans the PLC input registers and sets the output registers to the values determined by the program. When connected to the server, the input and output values stored in the PLC registers can be referenced, and the server can read and write to these memory addresses.

project

In Quick Builder, a working database file that enables you to make changes to the server database without affecting the configuration data that is currently being used to run the system.

project view

In Quick Builder, a window in which you can view, add, and modify any items in the current project file.

property

An attribute or characteristic of an object within the *Station Automation object model*. For example, a *display object* has properties that define its height, width and color.

property tab

In Quick Builder, a tab in the Project View window that displays information about the currently selected item or items. Most of the information displayed can be modified.

PV

Experion abbreviation for *process variable*.

PV algorithm

One of two types of algorithm you can assign to a point in order to perform additional processing to change point parameter values. A PV algorithm changes the value of the point process value (PV) input only. Contrast with *Action algorithm*.

PV clamp

For an analog point, a configuration that will immobilize the process value (PV) at 0% if it falls below the entry low limit value or at 100% if it goes above the entry high limit value.

PV period

An amount of time specified for the scanning of the point process value (PV) parameter. The PV period determines the frequency with which the scan will be performed by the server. The server groups point addresses into scan packets by PV period and controller.

Quick Builder

Quick Builder is a graphical tool that is used to define the hardware items and some point types in a Experion system. Quick Builder can run either on an Experion server, on another computer in your system, or on a laptop.

After defining hardware and points with Quick Builder, you download these definitions from Quick Builder to the Experion server database.

RCM

Recipe Control Module.

recipe

A set of points used in a process. The Recipe Manager option enables point parameters for

sets of points to be downloaded with pre-configured working values. The individual point parameters are the recipe 'ingredients.' Also known as *Activity Entity*.

recordset

An ADO object which contains data organized in fields and records.

redundant server

A second server used as a backup system. In a redundant server system the 'redundant' server is actively linked to the 'primary' server. Active linking ensures that data in the second server is constantly updated to mirror the primary server.

remote server

A server that supplies data to a local server via either a local area network (LAN) or a wide area network (WAN).

report

Information collected by the server database that is formatted for viewing. There are several pre-formatted reports, or the user can customize a report. Reports may be generated on demand or at scheduled intervals. Reports can be printed or displayed on a Station.

REX

Request to exit.

RFI

Radio Frequency Interference

RLSD

Receive Line Signal Detect

RTS/CTS

Request to send/clear to send

RTU

See *controller*.

S88/S95

A set of standards and terminology for batch control. For more information about the standard, go to [www.isa.org](http://www.isa.org).

SafeBrowse object

A SafeBrowse object is a Web browser specifically designed for use with *Station*. SafeBrowse includes appropriate security features that prevent users from displaying

unauthorized Web pages or other documents in Station.

scan

The technique used to read data from a controller. Scans are conducted for point parameters with source addresses (for example, PV, SP, OP, MD, An). Experion uses demand, exception, and periodic scanning techniques.

scan packet

A group of point parameter source addresses assembled by the server and used as the basic unit of server data acquisition. The server groups points into scan packets based on the controller address that they reference and the scan period defined.

scan period

The time interval that specifies the frequency at which the Experion server reads input values from the memory addresses of controllers. Scan periods are measured in seconds; a scan period of 120 seconds means that the server scans the controller once every 120 seconds.

scheduler

A facility used to schedule the control of a point on either a periodic or once-only basis.

SCM

Sequential Control Module.

script

A script is a mini-program that performs a specific task. Scripts use the *Station Automation object model* to control and interrogate *Station* and its *displays*.

security level

Access to Experion functions is limited by the security level that has been assigned to each operator. Experion has six security levels. An operator is assigned a security level and may perform functions at or below the security level that has been assigned to that operator.

server

The computer on which the Experion database software runs.

Server software

An umbrella term used to refer to the database software and server utilities installed on the Experion server computer.

server Station

A computer running both the Experion database (server) software and the Station software.

set point

The desired value of a process variable. Set point is a point parameter, whose value may be entered by the operator. The set point can be changed any number of times during a single process. The set point is represented in engineering units.

shape

A shape is a special type of *display object* that can be used in numerous *displays*.

Shapes can be used as 'clip-art' or as *shape sequences*.

shapelink

A shapelink is, in effect, a 'window' which always displays one *shape* of a *shape sequence*. For example, a shapelink representing a point's status displays the shape that corresponds to the current status.

shape sequence

A shape sequence is a set of related shapes that are used in conjunction with *shapelinks*. A shape sequences can be used to:

- Represent the status of a point (Each shape represents a particular status).
- Create an animation (Each shape is one 'frame' in the animation.)

SLC

Small Logic Controllers

SOE

Sequence of events

softkey

A softkey is a function key which, when pressed, performs an action specified in the configuration details for the current *display*.

SOR

Scope of responsibility.

SP

Experion abbreviation for *set point*.

SQL

Structured Query Language

Standard history

A type of history collection for a point that provides one-minute snapshots and the following

averages based on the one-minute snapshots:

- 6-minute averages
- 1-hour averages
- 8-hour averages
- 24-hour averages

### Station

The main operator interface to Experion. Stations can run either on a remote computer through a serial or LAN link, or on the server computer.

When Station is running on the Experion server computer, it is often referred to as a *server Station*.

### Station Automation object model

The Station Automation object model provides the programming interface through which *scripts* control *Station* and its *displays*.

### status point

A point type used to represent discrete or digital field values. The point can have input, output, and mode values. Input values can represent eight discrete states and cannot be changed by an operator. Up to three values can be read from up to three consecutive, discrete locations in the controller and thus can represent up to 8 states.

Output values can be used to control up to two consecutive discrete locations in a controller. Output values can be automatic or operator-defined.

Mode values apply to output values and determine whether or not the output value is operator-defined or automatic.

### supervisory control

The action of writing information to a controller. Experion enables both automatic and manual supervisory control. See *Mode*.

### task

A task is any of the standard server programs or an application program that can be invoked from a *display*.

### TCP/IP

Transmission Control Protocol/Internet Protocol. A standard network protocol.

### terminal server

A device on the local area network (LAN) that connects to a controller by way of a serial

connection and enables the controller to 'talk to' the Experion server on the LAN.

timer

A timer is a programming mechanism for running *scripts* at regular intervals in *Station*.

trend

A display in which changes in value over time of one or more point parameters are presented in a graphical manner.

UCM

Unit Control Module.

Unreasonable High and Unreasonable Low alarms

Alarms configured for an unreasonably high value and an unreasonably low value for the PV of an analog point.

User Scan Task controller

A server software option used to configure a server database table (called a 'user file') to act as a controller. The server interfaces with the user file rather than the actual device.

In this way you can write software to interface with the server and to communicate with devices that are connected to, but not supported by, the Experion server. The Experion server can then scan data from the user files into points configured on the User Scan Task controller and, for control, the Experion server can write point control data to the user file or a control queue.

USKB

Universal Station keyboard

USR

Unit Start Request

utility

Experion programs run from a command line to perform configuration and maintenance functions; for example, the **lisscn** utility.

virtual controller

See *User Scan Task controller*.

WCF

Windows Communication Foundation.

WINS

Windows Internet Name Service.

WWW

World Wide Web.

zone

A defined space either inside or outside that has at least one entry.

---

## Notices

### Trademarks

Experion®, PlantScape®, and SafeBrowse® are registered trademarks of Honeywell International, Inc.

### Other trademarks

Microsoft and SQL Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Trademarks that appear in this document are used only to the benefit of the trademark owner, with no intention of trademark infringement.

### Third-party licenses

This product may contain or be derived from materials, including software, of third parties. The third party materials may be subject to licenses, notices, restrictions and obligations imposed by the licensor. The licenses, notices, restrictions and obligations, if any, may be found in the materials accompanying the product, in the documents or files accompanying such third party materials, in a file named `third_party_licenses` on the media containing the product, or at <http://www.honeywell.com/ps/thirdpartylicenses>.

### Documentation feedback

You can find the most up-to-date documents on the Honeywell Process Solutions support website at:

<http://www.honeywellprocess.com/support>

If you have comments about Honeywell Process Solutions documentation, send your feedback to:

[hpsdocs@honeywell.com](mailto:hpsdocs@honeywell.com)

Use this email address to provide feedback, or to report errors and omissions in the documentation. For immediate help with a technical problem, contact your local Honeywell Technical Assistance Center (TAC).

### How to report a security vulnerability

For the purpose of submission, a security vulnerability is defined as a software defect or weakness that can be exploited to reduce the operational or security capabilities of the software.

Honeywell investigates all reports of security vulnerabilities affecting Honeywell products and services.

To report a potential security vulnerability against any Honeywell product, please follow the instructions at:

<https://honeywell.com/pages/vulnerabilityreporting.aspx>

Submit the requested information to Honeywell using one of the following methods:

- Send an email to [security@honeywell.com](mailto:security@honeywell.com).
- or
- Contact your local Honeywell Technical Assistance Center (TAC) listed in the “Support” section of this document.

## Support

For support, contact your local Honeywell Process Solutions Customer Contact Center (CCC). To find your local CCC visit the website, <https://www.honeywellprocess.com/en-US/contact-us/customer-support-contacts/Pages/default.aspx>.

## Training classes

Honeywell holds technical training classes that are taught by process control systems experts. For more information about these classes, contact your Honeywell representative, or see <http://www.automationcollege.com>.



DIAMOND  
PARTNER

# Honeywell

AUTHORIZED DISTRIBUTOR

## De Gids & Feldman BV

### The Netherlands

w w w . d g f g . n l



## De Gids & Feldman

INSTRUMENTATION & FILTRATION